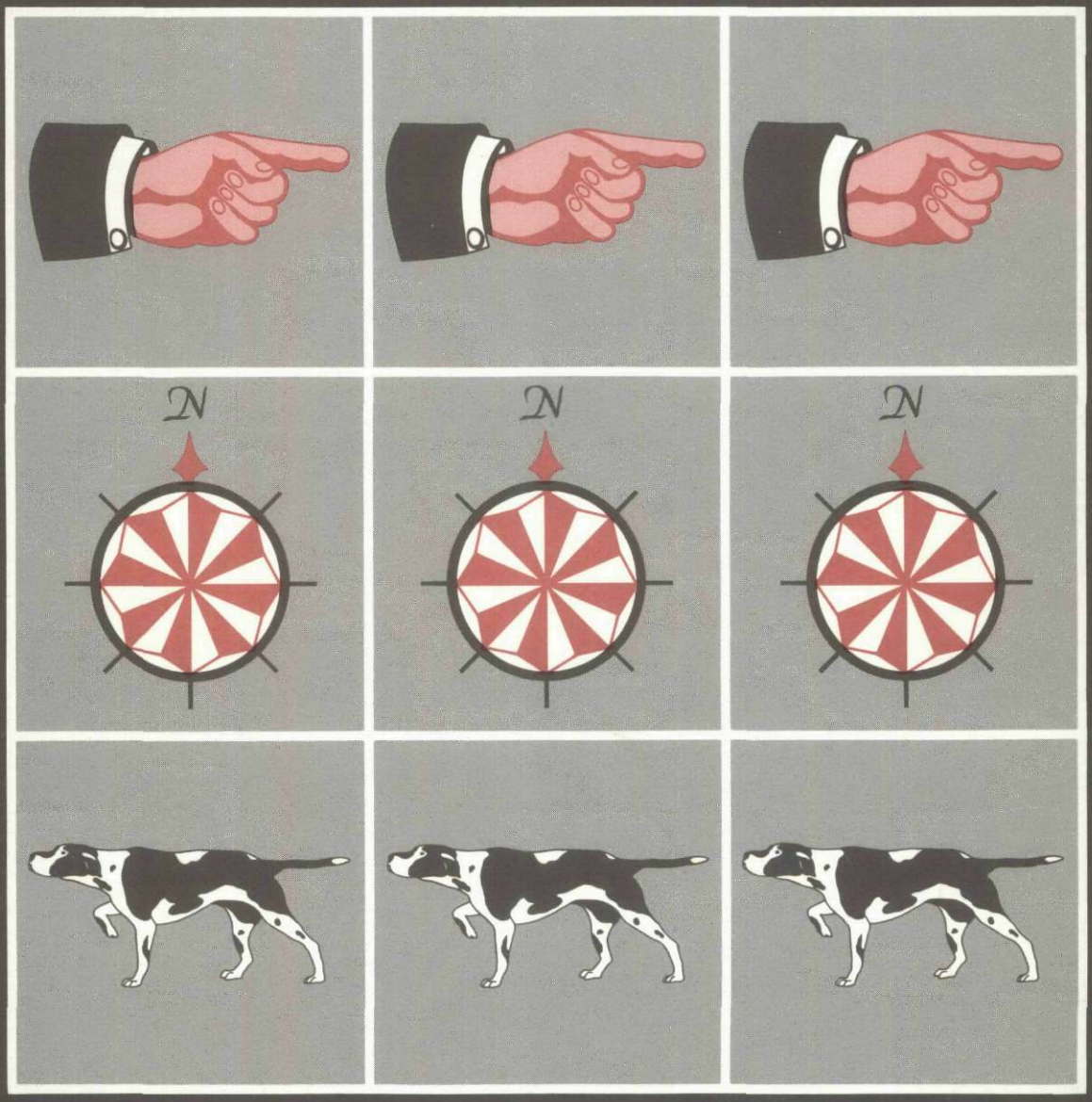
$4.00

programmer's journal

The Resource Journal for IBM PC Programmers

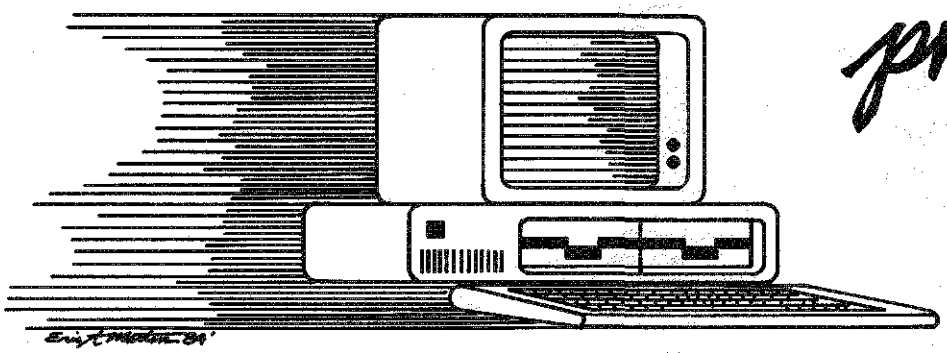
Link Up to Pascal's Pointer Power



Pointer Power!

Volume 2 Number 5

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



programmer's journal

CONTENTS

HOUSEKEEPING 2

LETTERS 3

PJ FORUM: THE 68000 AND 80286 from Dtack Grounded 4
Who will be running the show?

PJ PROFILE: DEVELOPING THE APX CORE EXECUTIVE 6
An interview with Paul Hunt and Danny Lepkoff of the A.E.C.

REDIRECTIONS: PC/IX, UNIX SUPPORT, FAT MAC by Frank D. Greco 10
Who is supporting what and why

C: THE C BANDWAGON by Dan Rollins 13
From assembler to C via a 216 byte .EXE main( )

BASIC: SCROLL CONTROL USING BASIC AND ASSEMBLER by Karl F. Thompson 17
Adding finess to your BASIC programs

ASM: SCREEN I/O ROUTINES by Joe Rayhawk and Stan Radzio 29
Faster screen handling with BASIC and ASM

BASIC: RANDOM ACCESS FILES (Part II) by Ronald Moak 36
The advantages of special header records

TOOLS: RTOOL, A PC-RESIDENT PROGRAMMER'S TOOLKIT review by Jim Gore 38
Have your development cake and eat it too

dBASE II: CALLING ASSEMBLY ROUTINES FROM d BASE II review by Warren Hillman 41
dBkey provides some much needed assembly modules

PASCAL: LINK UP TO PASCAL'S POINTER POWER by Chet Floyd 44
Straightforward use of Pascal's pointers

MISC 48

ISSN#0747-5861

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HOUSEKEEPING

Two monumental developments this month.

Brett Salter, of Peeks n' Pokes fame, has finished his long awaited debugger — Periscope. It is a full-featured symbolic debugger, system monitor and break-out switch for the PC or XT. The package includes a plug-in hardware board, manual and remote push-button. The 16K of memory on the plug-in board is write-protected so the resident portion of the debugger software sets up permanent shop. Imagine the functions your "ideal" debugger would have and Periscope probably has them, as well as some features you haven't thought of. Periscope does everything a normal debugger does with a number of wonderful enhancements. You can Go to a subroutine NAME, for example G @ VIDEO_IO, without having to hunt down its location in memory first. If your compiler generates line number references in the OBJ file, you can debug your program by source line number. Periscope permits the tracing of DOS and BIOS functions. Have you ever thought about how nice it would be to define an area of memory in a record/field format? Then, when you want to look at the PSP for example, it is unscrambled for you. Mr. Salter accepted suggestions for Periscope from some of the more prominent assembly language programmers on the IBM PC. For information contact:

Data Base Decisions
14 Bonnie Lane
Atlanta, GA 30328
(404) 256-3860

And now the BEST news. We have decided to follow the lead of Ashton-Tate and copy-protect Programmer's Journal. Yes, the first magazine on the block to take a stand on journalistic piracy. We KNOW some of you are passing PJ on to other programmers WHO DO NOT HAVE PAID SUBSCRIPTIONS. So with the next issue we will be printing PJ in a 3-D style type font that is illegible unless you have our special colored paper glasses. The glasses are photo-degradable and only last for two months. The page color format will become black ink on red paper to prevent Xeroxing.

This is, of course, a joke. But, if Prolocking becomes rampant, it would be disastrous. I thought we bought hard disks to stop the endless diskette swapping. Perhaps the floppy drive manufacturers who make drives that hold four or five diskettes at once are financially backing these new protection schemes that force you to have the original diskette in drive A:. As one programmer jokingly suggested, we could program a robot arm to swap Prolocked diskettes rather than buy a hard disk. This might be cheaper in the long run due to the current state of protection de-evolution. Soon recreational vehicle manufacturers will begin to base their mobile home prices upon your planned occupancy like motels do? Each family user of the RV can purchase a $500 laser-coded belt buckle that allows them to enter the vehicle. Why don't we "backup" the Prolockers into the nearest alley?

The IBM AT is finally here. Definitely a computer for "the rest of us" who have always been waiting for a software development machine.

— Greg □

PROGRAMMER'S JOURNAL

Published by Avant-Garde Publishing Corp.

Business Office:
Avant-Garde Publishing Corp.
1907 Garden Ave.
Eugene, OR 97403
(503) 345-3043

Editorial Office:
2765 Potter St.
Eugene, OR 97405
(503) 687-0398

Subscriptions and inquiries:
P.O. Box 30160, Eugene, OR 97403
(503) 345-3043

Programmer's Journal. Volume 2, Number 5.
Copyright © 1984 by Avant-Garde Publishing Corp.
Published bi-monthly.

Postmaster send address changes to:
Programmer's Journal
P.O. Box 30160
Eugene, OR 97403
(503) 345-3043

Publisher: Mary Carol Smith
Associate Publisher: Elizabeth Oakley
Editor: Greg Estes
Copy Editor: Joan Aschim
Production Manager: Christine Baldwin
Production Assistants: Marilyn Reaves, Jennifer Mullock
Cover Design: Marilyn Reaves
Illustrations: Julia King, Eric Martin

Contributing Writers:
Ron Moak
Frank Greco
Richard Larratt
Jim Gore
FNE
Robert E. Brown
Ron Burk
John W. Gowens
Warren Hillman

Subscriptions: Bi-monthly rates are $24 U.S.
($30 in Canada, and $40 elsewhere)

IBM and Personal Computer are trademarks of
International Business Machines, Armonk, New York;
dBASE II is a trademark of Ashton-Tate.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS

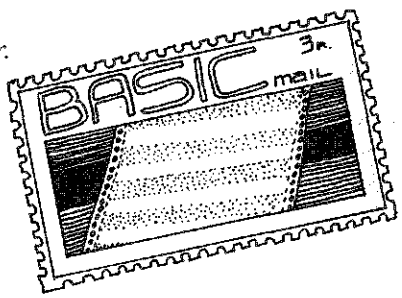
USEFUL LEADS

Please send me a copy of back issue Vol. 1 No. 3 to complete my file of Programmer's Journal.

A few comments in general. I value the material that you publish, especially the articles on assembler programming. The two-article series by Joshua Auerback was especially interesting to me — too bad you ended it in mid-sentence. The article by Wat Buchanon (Vol. 2, No. 3) was interesting, but marred by careless numbering of lines in the BASIC listings. Also I value the BASIC screen input routine ("Tools" by Greg Estes) in Vol.1 No.2. The items printed in the "Misc." section of the Journal are often very useful leads.

I am interested in purchasing a BASIC compiler. I would like to see a comparison of the features of the following BASIC compilers: IBM, Microsoft's BASIC and BUSINESS BASIC, and BETTER BASIC by Summit S.W.

Bernard H. Robinson, Jr.
DeBary, FL



CONTROL CODES

I have recently written the document "Control Codes for the Epson MX-80 (100) Printer with Graftrax-Plus". This document contains complete and accurate lists of Epson MX control codes which are arranged in both functional and numeric order. This presentation is unique and simplifies use of the codes and contributes to increased productivity. Additional material includes supporting text, programs, and a supplement covering the Epson RX and FX printers.

I would like to make this reference material available to other programmers who may also find it useful.

This package, "Epson Codes", may be obtained by sending $10 plus $1.50 shipping/handling to:

William Herron
Ardea Assoc.
4460 Del Mar Ave.
San Diego, CA 92017

RESET CORP

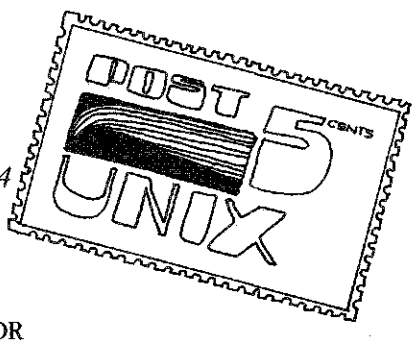
We recently began the sale of reset buttons for the IBM PC/XT computers. The principal advantage one gains with the use of our button is CONVENIENCE. One no longer needs to turn off the entire computer in order to reboot. The use of our button produces results similar to pressing Ctrl-Alt-Del. But, as you know, Ctrl-Alt-Del does not always work.

We have been testing this and a few other products for nearly a year and thought you might enter this information in your magazine under MISC. I am confident that many systems people will appreciate our button and its low cost. The button placement is in the back of the computer, either in the hole with the present knock-out plug, or through the XT access plate on the rear panel next to the power supply.

Very clear instructions are included. Programming tasks become less frustrating with the added convenience of a hardware reset button. Why didn't someone do it sooner? The cost is just $29.80 including mailing.

Darlene LaPointe
Minneapolis, MN

Reset Corporation
P. O. Box 14809
Minneapolis, MN 55414
(612) 489-5892



PROFESSIONAL EDITOR

I am a happy owner of the IBM Professional Editor except for one minor irritation. Whenever the editor is started, there is a three-second display of a "credits" screen. While I don't object to a display like that, it would certainly be nice if it could be shortened to one second or so. Does anyone know if there is a patch available to modify this characteristic of the program?

Additionally, is anyone collecting enhancements people to this editor, so we could submit them to IBM as a joint effort? We certainly would be willing to coordinate it provided you will let me put your name on the list of people requesting the enhancements. To kick it off, I suggest the following:

1. A set of shift commands similar to the IBM ISPF mainframe editor.
2. Split screen, even if we couldn't control the size of the split.
3. Cursor scrolling, again like the IBM ISPF mainframe editor.
4. True macros: i.e., a macro could contain a string of commands.

Bill E. Ferreira
P. O. Box 2633
Dublin, CA94568 □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EXCERPTS FROM DTACK GROUNDED THE 68000/16081 SYSTEMS JOURNAL, Issue #31,#32

by FNE, Copyright © 1984 by Digital Acoustics, Inc.

FOR THE UMPTEENTH TIME: The 68000 is clearly superior to the 80286 in some areas while the 80286 is clearly superior to the 68000 in other areas. By carefully selecting the application areas each company can — and has — produced detailed documents proving the clear superiority of their respective products. Each of these documents is technically accurate but necessarily limited in scope so as to restrict comparisons to those areas most favorable to their product.

The area in which the 68000 exhibits the clearest superiority to the 80286 happens to be our particular area of interest: large memories implying large programs and/or data arrays in single-user and mostly single-tasking environments. (If the situation were reversed this publication would bear the title "The Journal of Simple 80286/80287 Systems"!)

Please do not permit yourself to be confused by the fact that Motorola and Intel can each prove the superiority of their respective products.

Back to our Intel supporter's assertion that "the marketplace has spoken!". Boy has it ever! And we certainly agree that the marketplace seems to favor Intel-based microprocessors and their substantial 8080-based software support. However this is no recent development. Intel micros were favored when we started this newsletter and the situation has not changed in the three years which have passed since then.

As we asserted on page 2 of the last issue of DTACK GROUNDED, Intel dominance will continue (in our judgement) for about three more years. Emerging full software support for the 68000 and Intel segmentation woes in the midst of one-megabyte mass-market personal computer memories are then going to reverse the issue. (There are other contributing factors such as Intel's failure to match Motorola's— and Nat Semi's — mastery of CMOS. The 68020 will be built using 2-micron CMOS.) However today is NOT three years today hmmm?

So let us see what is happening today. That's easy; almost all companies which have chosen Intel micros are hurting and some are going out of business for lack of parts. This is especially true of those folks who are locked into the 80186. Intel simply cannot supply enough product to meet demand. One largish company which had been a high-tech high-flyer, Convergent Technologies, is being very very badly hurt by the shortage of 80186 chips.

In the meantime, IBM now owns 20% of Intel and seems to be having no problems getting microprocessors for the 2 to 3 million personal computers they are planning to ship this year. Intel spokespersons will look you in the eye and with a straight face assert that IBM does not get special treatment and has to stand in line for parts like everybody else. Ho ho ho! (Go ask the Convergent Technologies folks their opinion of that!)

So it turns out that our selection of a microprocessor vendor (Motorola) which is not the marketplace leader turned out to be a good idea after all. And since Apple is making lots of Macintoshes using BMHz 68000s then there will be no shortage of 68000L12's right? (Sigh) Wrong! Two weeks ago when we went out to buy another 100 L12's which had been available off the shelf until now, we were told that the factory (Motorola) was quoting 28 weeks!

What had happened to all the L12s? It turns out a couple of big outfits called IBM and XEROX had just bought up all the L12's between them. IBM is obviously stocking up for the release of its 9002 late this year. NOBODY can figure out what XEROX, a company which builds ever-slower and ever-lower-performance small computers, is doing buying L12's. Our guess is as imbedded controllers in Century Data Systems' hard disk drives. As you might suspect this certainly puts a kink in OUR plans to use L12's in our Grande board and elsewhere, yes?

If you will ignore the wicked grin we are displaying and the self-satisfied manner in which we are rubbing our hands together we would like to return your attention to one of those "busted tricycle" ads AMD was running a year ago. Remember the one which asserted contemptuously Motorola has their second sources spread out from here to Tokyo ? We are now buying Hitachi S12's which is what Hitachi calls their 12.5MHz 68000. In fact we now have more than $30,000 worth of 68000s on order! And you thought we were a small business? (We will return to Motorola when they get their delivery times back to something reasonable. Us round-eyes have got to stick together you know.)

THE SEARCH FOR THE GUILTY:

If some companies are in big big trouble over Intel's inability to produce enough parts then Intel is the dirty guy, right? No, that's not the way it works. Let us examine, for instance, the problem with delivery of the 8-bit single-chip Intel microcomputers. Three years ago hardly any personal-computer keyboards incorporated dedicated micros; now nearly all of them do. Two years ago the demand for these $3.50 chips went up by a factor of 3.5 and Intel largely kept production up

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

with demand. In the past year the demand has increased ANOTHER 5 times (!) and Intel has NOT been able to keep up with demand. Surprise?

In the case of the 80186 unforseen bugs in the design held up the start of mass production for a year. That delay apparently obscured from Intel the coming extraordinary popularity of the part which was foreseen by at least one outside observer. It is rumored that the pay cut and freeze instituted by Intel in Nov. '82 prompted the departure of the 80186 design team to greener (pun intended) pastures. Obviously, it is difficult to locate and eliminate bugs if there is no one around who knows how the chip works. What is REALLY scary is that apparently no one inside or outside Intel is certain that all the 80186 bugs are fixed even today.

A CONVERGENCE OF WOES:

The most visible 80186 customer who (like all other 80186 customers) is being hurt badly by the shortage of 80186s is Convergent Technologies. Convergent makes workstations for sale by OEMs. In 1983 nearly all of its sales were to Burroughs for a workstation designed around the 8086. During '83 Convergent designed a new workstation based on the 80186. They called this the "N-Gen" for "New Generation" and signed contracts not only with existing-customer Burroughs but also a whole bunch of other OEMs to deliver lots of "N-Gens" in 1984-85. Not only can they not get enough 80186s to even service Burroughs but it turns out they underestimated their costs on the "N-Gen" and are trying to retroactively negotiate a price increase (rumored to be 30%) with its OEMs. They have also tried without success to persuade Burroughs to accept revised "N-Gens" based on the 8086. [Shouldn't an 8086-based "N-Gen" be called an "O-Gen"?]

In the meantime the hush-hush $250 million deal they signed with AT&T turns out to NOT involve the N-Gen but a totally different design based on AT&T's (former) BELLMAC 32-bit micro and also on AT&T's 256K DRAMs. Jack Scanlon, AT&T's spokesman has just asserted that Convergent just plain is NOT going to get those chips to sell elsewhere as Convergent evidently had planned. (Just what in the hell was that top-secret $250 million pact supposed to cover?) In other words, not all of Convergent Technology's problems can be laid at Intel's doorstep.

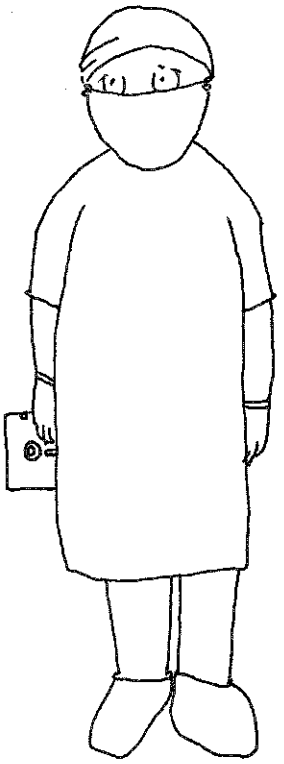
(We have just discovered that Convergent also has designed a graphics workstation called the GT around the 7220 and—surprise?—the Intel 80186. Congratulations guys!)

Intel — and Intel's customers — are victims of Intel's success! Back to that Intel supporter who wrote us that "the marketplace has overwhelmingly chosen the Intel line over Motorola's": that happens to be true for now but only IBM seems to be celebrating that fact even among Intel customers! (Regardless of protestations of innocence and purity on the part of Intel IBM is obviously going to continue to get all the parts it wants when it wants them. Here is how this works: Intel asserts that it is only delivering scheduled shipments. So if IBM needs more parts Intel can find an order dated Aug.'83 for those parts, wait for the ink to dry, and ship them! No? Do you also believe in the tooth fairy?)

THE FOLLOWING TRADEMARKS ARE ACKNOWLEDGED: Apple, II, II+, IIe, Applesoft, LISA: Apple Computer Co. Anybody else need a 166th million ack, have your legal beagles send us the usual threatening note. (To the best of our knowledge, Apple has not trademarked Macintosh yet. We don't have a THING planned for tomorrow; we just might mail something to the trademark registry office just for the hell of it. With trademarks, it's first come, first served.)

SUBSCRIPTIONS: Beginning with issue #19, subscriptions are $15 for 10 issues in the U.S. and Canada (U.S. funds), or $25 for 10 issues elsewhere. Make the check payable to DTACK GROUNDED. The address is:

DTACK GROUNDED 1415 E. MCFADDEN, STE. F
SANTA ANA, CA 92705
(714) 835-4884 □



Famous Computer Personalities #323: Dr. Laverne Sheed, MD, Inventor of the original Disk Operating System

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DEVELOPING APX CORE EXECUTIVE, A MULTI-TASKING MULTI-WINDOW ENVIRONMENT

An interview with Paul Hunt and Danny Lepkoff, software designers and principals in the Application Executive Corporation which produces and markets APX CORE.

APX CORE Executive is a multi-tasking, multi-window environment. Programs like dBase II, WordStar, and various compilers can run concurrently on an IBM PC using APX CORE. Data can be transcribed (sent) from one task to another, in either a manual or unattended fashion. Besides being an ideal software development tool, it allows you to integrate off the shelf software.

Paul Hunt and Danny Lepkoff, both shareholders and originators of the company, offered to share some of their difficulties and triumphs in the development process.

PJ: Why did you initially decide to do the APX CORE project? Creating a multi-tasking, multi-window program is quite a complex piece of work.

PAUL: Danny and I were the first two people to start the project as well as incorporate the company. We had both done development work on components of larger software systems, like data base managers, and screen I/O controllers. We found that there was not a product out there that would allow us to integrate existing modules, or complete programs, into a unified environment.

PJ: When did you first begin designing and working on your APX multi-tasking CORE system?

DANNY: Somewhere between two and three years ago.

PAUL: We incorporated in October of '82. So it is almost three years,now.

PJ: Did you write the multi-tasking part first or the windowing modules?

DANNY: We developed it in stages. We worked on the multi-tasking section first, the CORE of the system. We designed and implemented it initially.

PAUL: Some of the design features that we felt would be attractive to a number of different markets is the straightforward user front end. The Program Interface enables software writers to incorporate their product into APX CORE and access some of its resident functions. The features of APX are modular so they can be sold as a complete unit or separated into distinct units for various OEMs.

PJ: The Interface includes software hooks so a developer can control which concurrent task is active, which one is idle, or which one is in the foreground/background. Other hooks control the reading and writing of command keys, writing to a task's logical window, saving key strokes for later recall, or transcribing (sending) data from one task to another. All of this is set up via interrupt X'21' with a value of X'25' in register AH. Is this interface outlined in your regular manual?

DANNY: Yes, the calling parameters and register setups are outlined in the regular documentation. The data structures used to keep track of reading and writing to a logical screen, parameters for task control, and window blocks are illustrated in the text. Larger companies with programmers on staff, independent software developers, and OEM customers will appreciate the open access. This open integration of existing software packages has been missing. APX CORE is designed to fill this gap. We are trying to promote the idea of software packages being marketed as components of a larger system.

PJ: So if a company had a single-user accounting package and wanted to make it multi-tasking, they could use APX CORE as the key to such an implementation?

PAUL: Yes. There is a fair amount of flexibility that the developer can expect from APX.

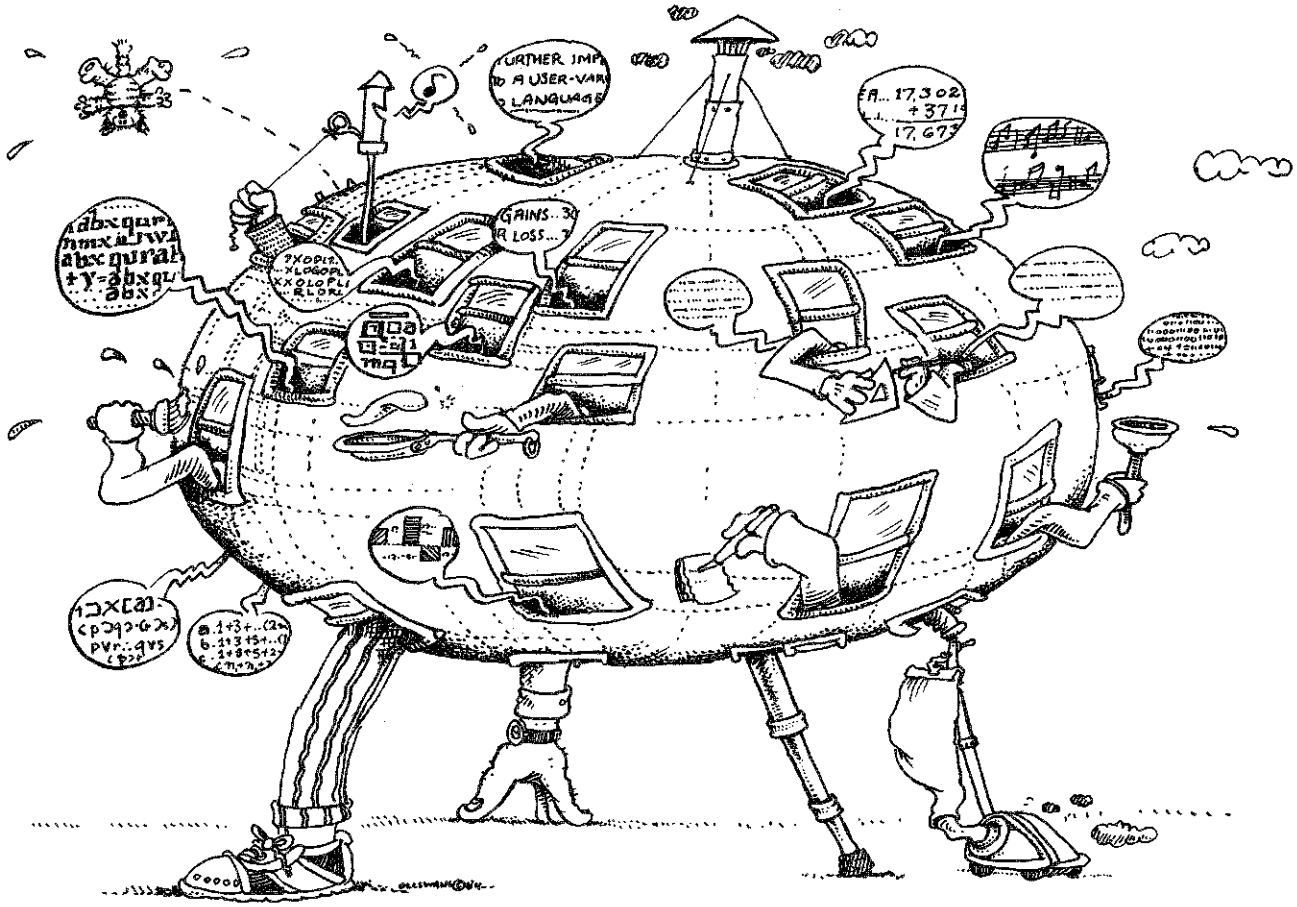
PJ: How does APX CORE allocate memory, both for itself and other tasks?

DANNY: There is a static part of the CORE that handles multiple tasks and it is loaded into high memory. If you had 640K it would start at the 600K boundary. Those are the resident routines. Memory is then divided, depending on how many tasks you have running.

PAUL: One technical consideration is worth mentioning here. APX CORE sets aside partitions for each task. We felt this was required in order to extend multi-tasking to most existing software without having to modify their programs. Other attempts, like Concurrent CP/M, present a kind of virtual memory facility, but on the 8088, with no memory management, the normal software must politely conform to such a paging system. There was no assurance that existing packages would follow new guidelines. The only option left was partitioning memory.

DANNY: To allocate memory for multiple tasks we have the user assign a ratio for how much memory one task should use in comparison to another. You indicate the number of tasks, say 3, followed by their memory usage ratio, say 321. This means that the first task uses 3 times as much memory as the last task, and the second task uses twice as much memory as the last task. The existing memory would then be divided

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



based upon the relative sizes of the tasks. This makes the environment more portable. If you take the system to another machine that has only half the memory, a proportional amount will still be allocated.

PJ: What were some of the major obstacles you had to overcome to complete your multi-tasking system?

DANNY: We had to learn the IBM PC down to the nitty-gritty. Aspects of the machine that were previously unknown to us had to be considered. The learning process was considerable. Certain software packages chose to access I/O devices directly rather than through the operating system. We not only had to have an interface around DOS, we also had to have one around the device drivers. We were aware of these two problems at the beginning. As time went on we found many programs that even skipped the device driver route and accessed the hardware directly. Memory-mapped programs are a prime example. Other programs write directly to a port rather than a driver. Interfaces like ours work best with programs that use drivers. This was and continues to be a consideration. Each partition that we create must be almost exactly like the machine itself, as much as we can make it.

PJ: In your documentation you specifically describe three categories of software packages: ones that are completely compatible with APX CORE, ones that are restricted in certain ways, and only a couple that are incompatible. For the restricted series of packages, like VEDIT, a memory-mapped text editor, you supply special RUN files that are loaded along with the program. What do the RUN files do?

PAUL: Certain RUN files were built for more complex applications. These files are just a list of parameters that outline the restrictions APX must be aware of, like not providing a logical screen for VEDIT because it writes directly to the screen buffer no matter what you do. A windowing function for VEDIT would be disabled. We actually have a monitor module that we use to detect these "rogue references" when they occur.

PJ: Could you mention a list of guidelines that programmers could follow so their software could easily run in a multi-tasking environment like APX CORE?

PAUL: Danny and I have identified several considerations that we feel would make a package a better component within a larger system.

DANNY: If the program only accesses services via the operating system, it of course would be highly compatible and easily portable. If you wish to use device drivers involved with vectors, it is at least compatible with IBM PC configurations. Services through these vectors and those through DOS are perfectly matched with us. More importantly, whatever product you may be creating, be sure and separate your I/O modules. The functions should be confined to a few routines, so, in effect, you build in your own interface to peripheral services. If the situation arises for a non-standard interface, you can just plug in another module. Sprinkling around many different video I/O instructions in your program is a poor choice.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PAUL: One of the best examples of this is Lotus. They actually have their I/O module in a separate file and it is not included into the program until 1-2-3 is loaded. They have a general program which is hardware configurable by way of a separate I/O module. This has given them very high compatibility.

We have just completed another package called an Application Kit. It is simply our version of the Lotus style I/O interface which then makes Lotus fully compatible with APX.

DANNY: We used our Program Interface services that are included in the APX package to construct the I/O module. This channels Lotus I/O to the correct region buffer for windowing purposes.

PAUL: Full background processing has been created for Lotus.

PJ: What is the price of the APX CORE Executive?

PAUL: It retails for $195.

PJ: Will most Ramdisks work with APX CORE?

DANNY: Yes, we maintain a high compatibility with almost all Ramdisks. If a resident module, like a Ramdisk, is loaded before APX, then the Ramdisk is available to every task active in the system. If your Ramdisk is loaded after APX in a separate partition, then the Ramdisk is only useable within that task.

PJ: You should document the multi-tasking problems you have encountered with most commercial software. This would give everyone, independent developers and companies, a set of guidelines to follow for avoiding conflicts. If someone can come up with a list indicating the "rogue references" that most existing packages make, it would be a valuable tool for anyone who wishes to interface with them.

PAUL: The interactive commands that a user has access to with APX for controlling the environment, are processed by another resident task. This task controller is readily accessible by anyone through the Program Interface, and allows you to create your own system message handler, or any other interactive feature. You have an easily customized user interface.

APX CORE ($195)
Application Executive Corp.
600 Broadway, Suite 4C
New York, NY 10012
(212) 226-6347 □

THE PROGRAMMER'S SHOP™

helps compare, evaluate, find products. Straight answers for serious programmers.

SERVICES
- Programmer's Referral List
- Compare Products
- Help find a Publisher
- Evaluation Literature free
- Dealer's Inquire
- Newsletter
- Rush Order
- Over 300 products
- BULLETIN BOARD - 7 PM to 7 AM 617-826-4086

PUBLIC DOMAIN Research - Free
6 months paid research gives you leverage, learning. We found, combined, added to the best. All run, have source in C or ASM. Order $150 + get one free: Database, Editors, Modems, MSDOS RAMdisks & utils, Games in C.

RECENT DISCOVERIES
SCIL - Manage versions, changes to source code, documentation. Minimize confusion, disk space. Interactive. CPM 80, MSDOS $349.

"C" LANGUAGE

	LIST PRICE	OUR PRICE
MSDOS: C86-8087, reliable	$395	call
Desmet with debugger	159	145
Instant C - Inter., fast, full	NA	500
Lattice 2.1 - improved	500	call
Microsoft C 2.x	500	349
Williams - NEW, debugger	500	call
CPM80' Ecosoft C-now solid, full	250	225
BDS C - solid value	150	125
MACINTOSH: Full, ASM	NA	385

Compare, evaluate, consider other Cs

BASIC

	ENVIRONMENT		
BASCOM-86 - MicroSoft	8086	395	279
BASIC Dev't System	PCDOS	79	72
BASICA Compiler - BetterBASIC - 640K	PCDOS	NA	185
CB-86 - DRI	CPM86	600	439
Prof. BASIC Compiler	PCDOS	345	325
MACINTOSH COMPILER with BASICA syntax	MAC	NA	325

Ask about ISAM, other addons for BASIC

FEATURES
XSHELL Adds, Intelligent Batch files and other UNIX-like features to MSDOS. $215.

PROLOG86 Interpreter for MSDOS includes tutorials, reference and good examples. Learn in first few hours. For Prototyping, Natural Language or AI. $125.

EDITORS Programming

BRIEF - Intuitive, flexible	PCDOS	NA	195
C Screen with source	8080/86	NA	75
FINAL WORD - for manuals	8080/86	300	215
MINCE - like EMACS	CPM, PCDOS	175	149
PMATE - powerful	CPM	195	175
	8086	225	195
VEDIT - full, liked	CPM, PCDOS	150	119
	8086	150	119

UNIX PC

COHERENT - for "C" users	PClike	$500	475
VENIX - "true V7" w/FTN	PClike	800	775
XENIX - "true S3" - rich	PC	1350	1295

Ask about run-times, applications, DOS compatibility, other alternatives. UNIX is a trademark of Bell Labs

LANGUAGE LIBRARIES

C Tools 1 - String, Screen	PCDOS	NA	115
C Tools 2 - OS Interface	MSDOS	NA	92
GRAPHICS: GraphiC-source	MSDOS	NA	195
HALO - fast, full	PCDOS	200	165
Greenleaf for C - full	MSDOS	NA	165
ISAM: C to dBASE-source	8086	150	140
BTRIEVE - many languages	MSDOS	245	215
Cindex + - no royalties	MSDOS	NA	395
PHACT - with C	MSDOS	NA	250
dBC - by Lattice	MSDOS	NA	250
PASCAL TOOLS - Blaise	PCDOS	NA	115
SCREEN: CView-source	PCDOS	NA	195
Databurst	MSDOS	225	215
PANEL-86-many languages	PCDOS	295	265
WINDOWS for C	PCDOS	NA	139

Ask about many others for FTN, BASIC, PASCAL, C-ISAM, Screen, Stat, Graphics.

FORTRAN

ENVIRONMENT	LIST PRICE	OUR PRICE
MS FORTRAN-86 - Impr.	MSDOS $350	$ 225
Intel Fortran-86	IBM PC NA	1400
DR Fortran-86 - full '77	8086 500	349
PolyFORTRAN-XREF, Xtract	PCDOS NA	165

OTHER PRODUCTS

Assembler & Tools - DRI	8086	200	159
CODESMITH-86 - debug	PCDOS	149	139
Disk Mechanic - rebuild	MSDOS	70	65
HS/FORTH	PCDOS	220	210
IQ LISP - full 1000K RAM	PCDOS	175	call
MBP Cobol-86 - fast	8086	750	695
MicroPROLOG	PCDOS	NA	285
Microshell improve CPM	8080	150	125
Microsoft MASM-86	MSDOS	100	85
MultiLink - Multitasking	PCDOS	295	265
PFIX-86 Debugger	MSDOS	195	169
PL/1-86	8086	750	495
PLINK-86 - overlays	8086	350	315
Polylibrarian - thorough	MSDOS	99	89
PolyMAKE	PCDOS	99	89
PROFILER-86 - easier	MSDOS	NA	125
PROFILER - flexible	MSDOS	NA	175
Programmers Tlkt w/source	8086	NA	135
Prolog-86-Learn, Experiment	MSDOS	NA	125
READ CPM86 from PCDOS	PCDOS	NA	55
READ PCDOS on an IBM PC	CPM86	NA	55
TRACE86 debugger ASM	MSDOS	125	115
X Shell - IF-THEN-ELSE	MSDOS	225	215

Note: All prices subject to change without notice. Mention this ad. Some prices are specials.

Ask about COD and POs.
All formats available. 0904

Call for a catalog, literature, and *solid value*

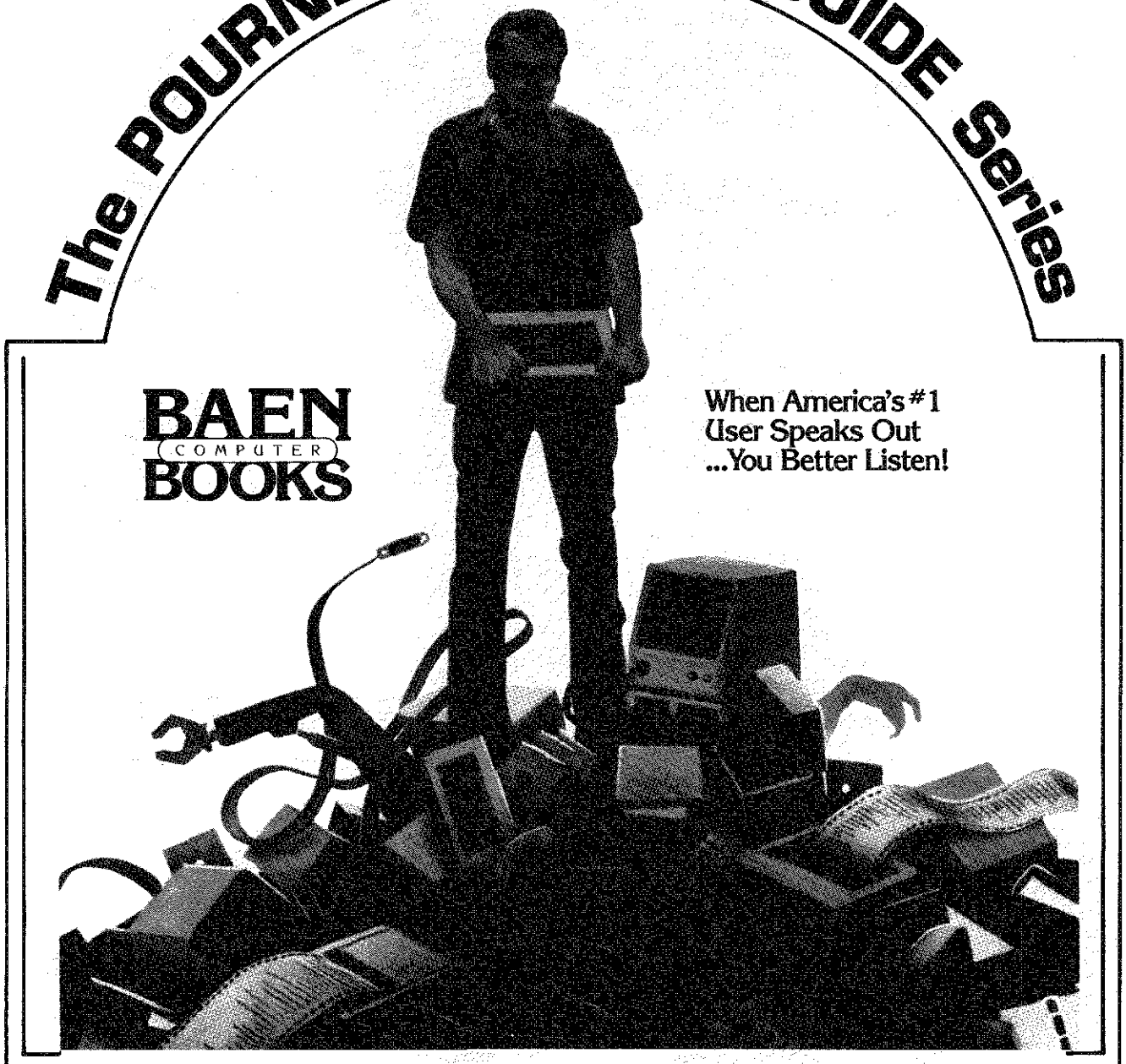
800-421-8006

THE PROGRAMMER'S SHOP™
128-J Rockland Street, Hanover, MA 02339.
Visa Mass: 800-442-8070 or 617-826-7531 MasterCard

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

INTRODUCING

The POURNELLE USERS GUIDE Series



BAEN COMPUTER BOOKS

When America's #1 User Speaks Out ...You Better Listen!

POURNELLE USERS GUIDE #1

THE USERS GUIDE To Small Computers

by Jerry Pournelle

55908-7 • 352 pp. • $7.95

Every POURNELLE USERS GUIDE title must be personally approved and in most cases will be commissioned or acquired by none other than Jerry Pournelle – #1 columnist in BYTE Magazine!

Also From Baen Computer Books in October–
THE SMALL BUSINESS COMPUTER TODAY AND TOMORROW
by William E. Grieb, Jr.
55907-9 · 288 pp. · $6.95

BAEN BOOKS DISTRIBUTED BY SIMON & SCHUSTER
MASS MERCHANDISE SALES COMPANY 1230 Avenue of the Americas New York, N.Y. 10020

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

REDIRECTIONS
PC/IX, UNIX SUPPORT, FAT MAC
WHO'S SUPPORTING WHAT AND WHY.

by Frank D. Greco, AGS Computers, Inc.

Unix is a multi-tasking multi-user operating system. Although IBM is only supporting their version of Unix for the PC, PC/IX, as a single-user environment, the multi-user code has not been removed. An IBM representative at a local Unix user group meeting said that prospective users could access it, but it is NOT supported by IBM. This was due to the "additional licensing charges levied by AT&T for multi-user capability which IBM did not want to pay". Dubious I say. If you believe that IBM did not want to charge $1000 for a multi-user version of PC/IX instead of the $900 for the current single-user version because of additional AT&T licensing fees, think again. IBM does not want to promote a sluggish multi-user 8088 based system that has minimal security.

In any event, this raises an important issue. If a licensed IBM PC/IX single-user runs the normal Unix multi-user code that exists in PC/IX; is he or she breaking the law? There have been several magazine articles stating the simplicity of hooking up another terminal to the PC when using PC/IX, but none of them mention the legality issue. Would AT&T want to prosecute PC/IX users running in a multi-user mode and thus involve IBM in a legal tete-a-tete? That is not likely but not entirely impossible. That would definitely be a strategic move on AT&T's part to reduce the possibility of IBM's PC/IX becoming an industry standard instead of AT&T's System V.

Here are some interesting notes for program developers investigating PC/IX. It includes several enhancements done by the University of California at Berkeley (BSD) Unix development group. The terminal description database "termcap" is included along with a higher level C language library of terminal screen I/O routines "curses". However the popular, powerful, and although occasionally cryptic screen editor "vi" (pronounced "vee eye") is not distributed with PC/IX. One reason is that PC/IX is based on the Intel SMALL memory model (64K bytes Data and 64K bytes Instruction Segment size maximums). "Vi" is a large program, typically over 128K on most systems. So either IBM or Interactive Systems hack "vi" (both usages of "hack" !) to fit inside 64K, OR they promote another screen editor. IBM opted to use Interactive's INed editor, which happens to be an excellent editor for the single-user PC/IX. I understand, however, that INed causes the PC to crash when PC/IX is run multi-user.

HI HO SILVER LINING
-or-
IS IBM REALLY PROMOTING PC/IX?

When I wanted to know more about PC/IX I called IBM. As most of you developers out there would concur, this is a logical, although non-intelligent act. From past experiences of dealing with IBM PC support, I had a feeling about the response from IBM regarding their version of Unix. Would you believe:

"PC/IX? is that a utility program?"

"PC/IX? you mean the IX system for the PC?"

"What is PC/IX?"

"PC/IX? Oh yes, that's the UNEX (sic) programming language."

These are actual responses that I received from the resident system analysts at several IBM Product Centers. I kid you not.

Now I am not picking on IBM system analysts, but if IBM was so interested in the Unix market...wouldn't you think that the higher ups would disseminate PC/IX product information more rapidly through the ranks? Didn't IBM announce PC/IX in January? We are almost in the third quarter of 1984. It would seem that perhaps PC/IX was almost ENTIRELY a strategic move on IBM's part to steal some attention from AT&T's 3B series of machines. PC/IX received almost as much attention in the press as the AT&T entrance into the commercial computer marketplace. The impact of AT&T's announcement was severely lessened and consequently IBM's PC dominance and IBM's Unix involvement was not even impacted.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Another important rumor is that IBM is currently porting BSD 4.2 (Berkeley) Unix to the National Semiconductor 16032 microprocessor. BSD Unix (Berkeley Standard Distribution) is an extremely popular version of Unix that incorporates virtual memory, high-level file/record locking, and advanced networking features; Unix System V from AT&T does not contain most of these features. Also, BSD Unix development is heavily tied to DEC's VAX architecture; the 16032 is a very VAX-like microprocessor.

Besides the System III 8088 PC/IX and the BSD 4.2 16032 port, there are rumors surrounding IBM and the Coherent OS from Mark Williams. Even if this Coherent rumor is truly a rumor, it is starting to make its way into several high visibility trade mags. IBM is known for starting rumors (both true and false) for strategic purposes. Another very interesting thought regarding IBM's stance on Unix now starts to coalesce. Suppose IBM encourages several varieties of Unix on their machines and purposely increases the Unix confusion among MIS managers. After helping non-AT&T versions reach equity with Mother Bell's home version, out the BLUE (sorry...) comes IBM with their Unix Superset. LOOKS like Unix, ACTS like Unix but it is NOT Unix. It even has MS-DOS/PC-DOS emulation. Developed internally without AT&T's explicit help (thanks to the $43,000 Unix source code lesson...err...license) and distributed with substantial IBM programming support, it might be the silver lining that most programming shops are looking for amongst the Unix clouds. After all, it has those three magical letters.

MACINTOSH NEWS

At a recent Macintosh developer's meeting in New York City, I was told that the Lisa would be phased out in two to three years and a new high end micro would be introduced. It's code name is "FAT MAC" and it incorporates the same Xerox technology that made its predecessors famous plus some goodies: super high resolution COLOR graphic (according to an Apple spokesman, the Mac's ROM ALREADY supports color), enhanced multi-channel sound system, high density 3 1/2 inch Winchester disks and it will use the 68020 microprocessor. This processor is the full 32 bit/virtual memory version of the 68000. Remember folks, the 68000 is a 16 bit chip with 32 bit registers. This new MAC development was spurred by Steve Jobs' visit to Japan where he saw some new Japanese micros that incorporated Xerox technology. Apparently he was concerned enough to get Burrell Smith, the designer of the MAC's elegant digital board, to start working on "Fat MAC".

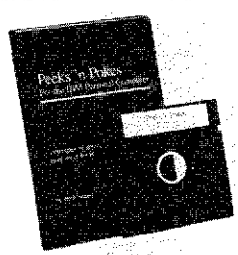
Most Macintoshes internally used at Apple and all of the Macintoshes at Lotus Development have the 512K Macs. An Apple spokesman said the memory upgrade from 128K to 512K will be available in December. Assuming a price of $50.00 per 256K chip times 16 (the Mac has no parity chip) and some labor costs, the cost of upgrade could be about $1000. Upgrading a IBM PC from 128K to 512K costs about $500. Also the boards that are taken out of the Mac will be recycled in another Apple product. What Apple product you say? Well the Apple spokesman would not elaborate.

Major Coup for Apple: Hiring Alan Kay from Atari. Kay is the long time Xerox PARC Developer that was heavily involved in the Smalltalk programming environment, i.e., windows, mice, bit-mapped graphics, programming environments...etc. One of Kay's favorite pet projects was his Dynabook project, which in a nutshell was a lap-based version of a extremely usable machine like the Macintosh. It never saw any commercial interest then (1970's) but Kay has been quoted as saying that all the Dynabook needed was a powerful microprocessor like the 68000 in CMOS. Perhaps at Apple, Alan Kay might see his idea come to fruition.

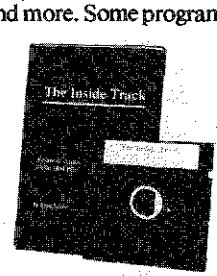
Know Thy PC!

Are you writing programs in BASIC or Pascal? The popular **Peeks 'n Pokes** has a disk with 58 programs and a 38-page manual that helps you get 'underneath the covers' of the PC. Learn how to use PEEK, POKE, INP, OUT, and DOS/BIOS function calls to do what you want, fast! Do you want to perform functions not available from BASIC or Pascal? It's all explained in the manual and demonstrated in the sample programs. Source code included!

Peeks 'n Pokes shows you how to:
- Access the system's configuration
- Unprotect BASIC programs
- Scroll part or all of the screen
- Access the file directory
- Logically swap printers
- Read and change the keyboard
- Find more Peeks and Pokes
- And much more...for only $30.00



Want to know more? **The Inside Track!** is a collection of advanced utilities for the PC programmer. It contains a disk with 61 programs, a 42-page manual, and a fold-out memory map that helps you get better performance from the PC. With this package you can give your programs assembler-assisted speed from high-level languages, get control over memory, customize and control the PC, and more. Some programs require DOS 2.00. Source code included!



The Inside Track! shows you how to:
- Read/write files as fast as DOS
- Display data on the screen faster
- Reserve memory for your use
- Copy memory to another location
- Copy-protect your programs
- Load large programs faster
- Control the keyboard settings
- And much more...for only $45.00

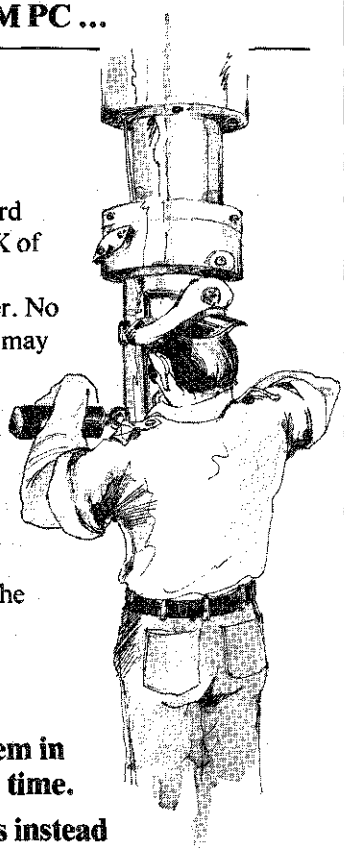
MasterCard and VISA accepted. Shipping charges: $2.50 per order for UPS; $2.50 per item for First Class Mail to USA and Canada; $6.00 per item for Air Mail outside USA and Canada. Dealer inquiries invited.

Data Base Decisions • 14 Bonnie Lane • Atlanta, GA 30328 • 404/256-3860

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Debugging Bugging You?

Torpedo program crashes and debugging delays with debugging dynamite for the IBM PC ...



UP PERISCOPE!

First, you install the hardware.

The hardware's a special memory board that fits in a PC expansion slot. Its 16K of write-protected memory contains Periscope's resident symbolic debugger. No runaway program, however berserk it may be, can touch this memory!

Then you UP PERISCOPE.

Use Periscope's push-button break-out switch to interrupt a running program ... even when the system's hung! Periscope supports Assembly, BASIC, C and Pascal. In addition to the usual debugging capabilities, some of Periscope's features are:

Stop your system in its tracks at any time.

Use symbol names instead of addresses.

Run a program on one monitor and debug on another.

Monitor your program's execution with Periscope's comprehensive breakpoints.

Debug memory-resident programs.

Put your time to better use.

The Periscope system is $295. It carries a 30-day money-back guarantee and includes the memory board, remote break-out switch, debugger software, 100-page manual, and quick-reference card. The memory board is warranted for one year. A demonstration disk is $5.00.

System requirements for Periscope are an IBM PC, XT or Compaq, PC-DOS, 64K RAM, 1 disk drive and an 80-column monitor. For MasterCard and Visa orders only, call 800/421-5300 (ext. R96) 24 hours a day. For additional information, call 404/256-3860 from 9 AM to 5 PM Eastern Time.

Get your programs up and running;

UP PERISCOPE!

Data Base Decisions / 14 Bonnie Lane /Atlanta, GA 30328

PINK NOISE DEPARTMENT

Now that RCA has dropped out of the video disk player market, look for closeout prices on their video disk players. I have seen prices around $100 for them.

More details on Commodore's Z8000 Unix machine: it comes with 256K RAM, 128K bytes of screen RAM, 32K of system code in ROM, two 1.3 Megabyte 5 1/4 inch floppies, serial and parallel ports and super high resolution graphics. On special hi-res monitors, the Commodore machine can display 1024 X 1024 pixels in 8 colors. The operating system as mentioned before is the Coherent Unix-like operating system from Mark Williams. Commodore renamed the system software "CO-HOS"; another name for a Unix-based operating system? ...why buck the trend at this stage. Accessories for this machine include an IEEE-488 interface (one of Commodore's favorite interfaces), a Z8070 math co-processor, RAM up to 16 Megabytes (with the help of some programming pyrotechnics), and a 3-inch Winchester drive (from Rodime). Commodore has upped the suggested price to around $4000. I hope this machine sees the light of day and that the recent management shuffling at Commodore doesn't affect the introduction of this machine.

Hayden Book Company has started to offer a video tape series dedicated to the Unix Operating System. The series is called the AT&T Bell Laboratories' Software Technology Videotape Series. Featured are founding fathers Ken Thompson, Dennis Ritchie, and Brian Kernighan. The first two tapes are called respectively, "Making Computers Easier to Use" and "Making Computers More Productive" both in BETA, VHS or 3/4-inch formats. You can reach Hayden at 1-800-631-0856.

Since the Epson LQ printer is receiving very mediocre reviews lately, the Japanese peripheral company is pushing up the announcement of their ink-jet printer for late 2nd quarter or 3rd quarter of 1984. There are very strong rumors that it will use a 24 nozzle (equivalent of 24 pin) printhead and print at 200 cps in draft mode and 80 cps in letter-quality mode. And like other ink-jets (like the Hewlett Packard $500 model), it will be whisper quiet. Someone at Epson suggested a price of "around $1900" when I spoke with them. Other rumors in the copier field have Epson combating laser printer vendors with an extremely low priced model which will be due out in the U.S. in the 4th quarter of 1984. This would make for good market timing, because Apple is making their (supposedly OEMed from Canon) laser printer available for the MAC and Lisa at the end of 1984. Articles on laser printers will proliferate at that time and Epson will benefit from the free PR. Don't be surprised if Epson isn't the only one to come out with a laser printer then. □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

THE C BANDWAGON

by Dan Rollins

I've recently climbed on the C bandwagon. I'm finding that C is an ideal alternative for frustrated assembly language programmers. I was able to write a directory search program (to search a path of directories) in a few hours (this was my introduction to recursive procedures). I might have written the same program in assembler with maybe a dozen times the amount of effort, but it would not have had as many bells and whistles, nor would it be easy to modify.
The beauty of C is that there is such a massive library of functions that are already written, and maintaining a library is a snap. Also, C is intriguing — it's fun to learn.

On the other hand, there are several things that really bug me about C. First, programs are quite difficult to debug — far harder than BASIC, and only marginally easier than assembly language. A good symbolic debugger would come in handy, but the edit/compile/link process itself is very time-consuming. Why are all C compilers written in C? Imagine a BASIC interpreter written in BASIC! If Borland Int'l ever comes out with Turbo-C, I'll be the first to buy it. A C interpreter would be great, but the only one I've heard of costs far too much. C would be enormously popular as a low-cost interpreted language.

A serious drawback of C is the way that pointers are handled. As an assembly language programmer, I work with pointers in every program — I'm very comfortable with the concept of pointers. But I had a tough time getting a feel for C pointers. I think that part of the problem is the way they are declared in the source code:

```
char c, *s;
```

looks as if it declares two items of type "char". It seems to me that a declaration like:

```
char c;
ptr  char s;
```

would declare a pointer more clearly.

Here's another concept that gave me trouble. I could glean no meaning from a line like:

```
while ((c=getchar()) != EOF)
```

How can you compare an assignment statement to anything? It's just another C quirk that I had to learn by rote — that an assignment, when treated as a value, takes on the value of the lvalue (that's another innane term) that is on the left side of the equal sign.

I had a devil of a time trying to figure out what was wrong with a program that was supposed to display a "business graphics" box. I had to disassemble the "putch" library function to find that it (and other character output functions) will often shave the high bit from a byte. I really can't think of any reason to do this and it's not discussed in the "machine dependent" portion of the Microsoft C reference manual. Of course I immediately excised all such idiotic restrictions by placing my own version of "putch" in the MS libraries.

Finally, I resent the loss of control that is inherent when moving from assembly language to any compiler. Where are my segments? What are the values of the data that DOS passes in the all-important PSP? I also resent the large size of any C program. I like to write programs in 200 or 300 bytes, not the minimum 12,000-byte EXE files produced by the Microsoft C compiler. Fortunately, it is possible to solve this problem.

My solution was to write my own version of the front-end module — the module to which DOS passes control after the C program is loaded. My goal was to eliminate unneeded overhead code from the object file and to be able to create a COM-format file from the result.

Listing 1, COM_C.ASM is a substitute for CS.OBJ, which is linked to the front of each "small model" program compiled with Microsoft C. The tricky part is organizing the segments into a single group so that the resulting EXE file can be converted into a COM file. I also needed to make sure that the segment registers point to the correct place. From a COM format C program, it is only a short step to writing installable device drivers and memory-resident interrupt handlers in C.

Using the COM_C module, or your own adaptation, can have have impressive consequences. The minimum small module C program,

```
main(){}
```

written with undoctored Microsoft C creates an EXE file that is 8192 bytes long. Using COM_C, you can write programs that start as small as 216 bytes. The library functions tend to pull in a lot of code, but this too, can be minimized. The following C programs are functionally identical, but the difference in size is dramatic:

```
_main(args)             /* COM file is 4109 bytes */
char *arg
{ cprintf("This is the input string:'%s'\n",args);
}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
main(args)              /* COM file is 595 bytes */
char *args
{ cputs("This is the input string: '"):
  cputs(args):
  cputs ("'\n"):
}
```

PROGRAMMING ENVIRONMENT

For useful programming tools, I offer one batch file and this description of my working environment.

I use several PC and XT configurations at the different places I work. My favorite is the one I use at my home/office. I have a PC with two diskette drives and a 15Mbyte Tandon external hard disk. I have 576K of memory, and I use the Talltree JBOOT and JDRIVE to configure my machine with 256K for programs and 320K for a drive D RAM-disk.

My AUTOEXEC.BAT copies my editor (the IBM Personal Editor) and its satellite files to drive D, and sets a PATH that includes a UTILS sub-directory for all DOS external commands and other often-used utilities.

I have discovered two important secrets for working with a hierarchy of directories. First, I set my DOS prompt like this:

```
PROMPT $p$g
```

or

```
PROMPT ESC[1m$p$gESC[m
```

This lets me know which sub-directory is the current default. The latter version only works when the ANSII.SYS driver has been installed (Note: ESC is an ASCII 27 so this version must be entered from a batch file). It makes the DOS prompt be displayed in high-intensity. The PROMPT command changes this:

```
C>_
```

to this:

```
C:\wp\book\chap3>_
```

My second handy hint is to use the RAM-disk as a way to get around the path-less filenames needed by DOS 1.10 software. First, I select my directory for the hard disk, then I set drive D as the default. That makes a command like:

```
D:\>EDIT c:list1.txt
```

work the same as:

```
D:\>EDIT c:\wp\book\chap3\list1.txt
```

which would normally cause an error. I still must exit the editor and use CD to access a different directory, but this is very fast (the editor being on RAM-disk), and I don't need copies of the editor's satellite files on every directory.

Another advantage of a RAM-disk is more obvious, but many people miss it. When you compile a program, the compiler reads a chunk of the source file, processes it, and outputs a chunk of the object file. This equates to a lot of disk head movement as the compiler juggles the two files. I will often use the RAM-disk for both the input and output files, but it is almost as fast to process the source file from physical media — as long as the object file is sent to the RAM-disk (the OBJ file is often a throw-away anyway). This has the added advantage of keeping the source file safe from power failure and the other peculiar things that tend to happen at 3 AM.

Listing 2 is a batch file I use to compile a COM format C program. It takes maximum advantage of a large RAM disk, going to the physical disk only to save the new version of the source code. It also automates the edit/compile/link cycle to simplify my life. Of course the batch file itself should be on RAM-disk for maximum throughput.

C Programmers: Program three times faster with *Instant-C*™

Instant-C™ is an optimizing **interpreter for C** that makes programming three or more times faster. It eliminates the time wasted by compilers. Many repetitive tasks are automated to make programming less tedious.

- Two seconds elapsed time from completion of editing to execution.
- **Symbolic debugging;** single step by statement.
- Compiled execution speed; 40 times faster than interpreted Basic.
- Full-screen editor integrated with compiler; compile errors set cursor to trouble spot.
- Directly generates .EXE or .CMD files.
- Follows K & R—works with existing programs. Comprehensive standard C library with source.
- Integrated package; nothing else needed.
- Works under PC-DOS*, MS-DOS*, CP/M-86*.

More productivity, less frustration, better programs. ***Instant-C***™ is $500. Call or write for more info.

Rational Systems, Inc.
(617) 653-6194
P.O. Box 480
Natick, Mass. 01760

Trademarks: MS-DOS (Microsoft Corp.), PC-DOS (IBM), CP/M-86 (Digital Research, Inc.), Instant-C (Rational Systems, Inc.)

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;-----------------------------------------------------------------------
; COM_C.ASM              By Dan Rollins              6-15-84
;
;
; Creates a COM-format file from a Microsoft C program.
;
; Compile the C program using these steps:
;
;    MC1 my_prog -mS
;    MC2 my_prog           /* use -v to save 10 bytes on each user function */
;    LINK com_c+MY_PROG,MY_PROG,con,mcs;    /* ignore "No Stack" warning */
;    EXE2BIN MY_PROG MYPROG.COM
;    ERASE MY_PROG.EXE
;
; Notes:
;
; The resulting C program MUST be run under MS-DOS or PC-DOS version 2.x
;
; To obtain the most concise code file:
;
; * Link with TINYMAIN.C (instead of the library "_main")
;   (if you don't use ARGC and ARGV, name your own "main" function as "_main".
;    "_main" receives 1 argument, a pointer to the input string)
; * Use console functions like "cprintf" (instead of "printf")
;   (use to "cgets", etc. for maximum brevity)
; * Use the "_exit" function (instead of "exit")
; * Write your own file-handling routines using the "intdos" library function
;

com_group group prog,data,zzzz

prog segment para public 'prog'  ;make PROG the first segment linked
prog ends

data        segment byte public 'data'
            public  _ver, _top, _base, _psp, _mbase, _mnext, _msize

data_start label byte               ;addr used to locate this internal segment
null        dw    0
_ver        db    "Lattice C 2.00"
_top        dw    0                  ; top of stack (relative to SS)
_base       dw    0                  ; base of stack (relative to DS)
_psp        dd    0                  ; program segment prefix pointer
_mbase      dw    0                  ; base of memory pool
_mnext      dw    0                  ; next available memory location
_msize      dd    0                  ; number of bytes left in pool
_env        dd    0                  ; pointer to environment

ovferr_msg db    0dh,0ah,"** Stack overflow error **",0dh,0ah,7,"$"
data  ends

zzzz segment byte 'data'
stack_base label byte
zzzz ends

-----------------------------------------------------------------------
```

```
prog    segment para public 'prog'
        assume cs:com_group, ds:com_group, es:com_group


        public c, xcexit, _exit
        extrn _main:near


        org     100h                    ; for COM format programs
c       proc    near

;--- set up external variables

        mov     word ptr _psp+2,ds
        mov     ax,word ptr ds:[2ch]  ;DOS "environment" begins at paragraph
        mov     word ptr _env+2,ax    ;named at PSP:2ch...save it

        mov     bp,sp                 ;setup BP for call to main
        mov     _top,bp               ;_TOP variable is highest addr in stack

        mov     _base,offset com_group:stack_base ;_BASE is bottom of stack

        mov     ax,_base
        mov     _mbase,ax             ;these variables are used by
        mov     _mnext,ax             ; memory allocation functions

        mov     bx,_top
        sub     bx,ax
        mov     word ptr _msize,bx

;----- find where the DATA segment starts within the COM_GROUP

        mov     ax,offset com_group:data_start
        shr     ax,1
        shr     ax,1                ;since DATA segment is paragraph-aligned
        shr     ax,1                ; (within COM_GROUP), we can find the
        shr     ax,1                ; paragraph address by adding its offset,
        mov     bx,cs               ; divided by 16, to the PSP paragraph
        add     ax,bx

        mov     ds,ax               ;now points to paragraph of DATA segment
        mov     es,ax               ;DS, ES, and SS
        mov     ss,ax               ;all point to C data area

        mov     bp,sp               ;save frame before stacking argument string
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;-- now copy the DOS command line onto the stack for the "_main" function

        mov     si,81h                  ; point to first character (its in CS)

        mov     cl,byte ptr cs:[80h]    ;length of command line
        xor     ch,ch


        mov     bx,cx                   ;get length of input +3 (2 bytes
        add     bx,3                    ; for "c" +1 for rounding)
        and     bx,0fffeh               ;force even number of bytes
        sub     sp,bx                   ;allocate space on stack
        mov     di,sp
        mov     byte ptr ss:[di],'c'    ;store dummy program name
        inc     di
        jcxz    m_2                     ;skip if no bytes to move
m_1:                                    ;-- move bytes to stack
        lods    byte ptr cs:[si]        ; get input character SI++, CX--
        mov     ss:[di],al              ; place in buffer on stack
        inc     di
        loop    m_1
m_2:
        mov     byte ptr ss:[di],0      ;append null terminator byte


        mov     ax,sp
        push    ax                  ;pointer to string for "_main"
        call    _main               ;invoke program that invokes user program
        mov     ax,4ch              ;exit to PC-DOS, returning ERRORLEVEL
        int     21h                 ; code of 0 (normal completion)
c       endp


xcexit  label   far
_exit   proc    near
        mov     bp,sp
        mov     ax,[bp+2]           ;GET exit code (for ERRORLEVEL)
        mov     ah,4ch
        int     21h                 ;DOS EXIT service
_exit   endp



;---------
; XCOVF
;       Executed when the invocation of a C function causes stack overflow
;       Needed even if -v option is used in the MC2 code generation pass
;       because the library routines need this address.

        public  xcovf
xcovf   proc    far
        mov     dx,offset data:ovferr_msg
        mov     ah,9
        int     21h                 ; generate error message
        mov     ax,4c01h
        int     21h                 ; exit with ERRORLEVEL code of 1
xcovf   endp

prog    ends
        end     c
```

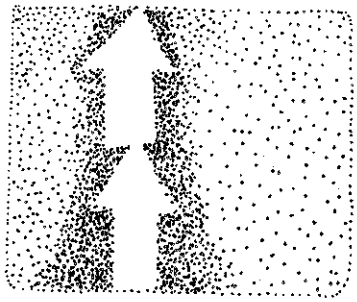
Listing 2:

```
. === MCC.BAT compiles a Microsoft C "small model" program ==
. === into a COM file ===

if NOT .%1. == .. goto cont1
echo It must be late...You should specify a filename (no extension)
goto exit

:cont1
. === setting up the ROM-disk ===
if not exist mc1.exe      copy c:\mc\mc1.exe
if not exist mc2.exe      copy c:\mc\mc2.exe
if not exist tinymain.obj copy c:\mc\tinymain.obj
if not exist com_c.obj    copy c:\mc\com_c.obj
if not exist mcs.lib      copy c:\mc\mcs.lib

mc1 %1 -mS
if exist %1.q goto cont2
pause  Errors found ... about to re-edit
edit %1.c
mcc %1

:cont2
. === copying source code to hard disk ===
copy %1.c c:
mc2 %1 -v
link com_c+ %1+ tinymain,%1,nul,mcs;
exe2bin %1 %1.com
del %1.exe
:exit
. === MCC.BAT jcl completed ===  □
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BASIC: SCROLL CONTROL USING BASIC AND ASSEMBLER

ADDING FINESSE TO YOUR BASIC PROGRAMS



by Karl F. Thompson

Did you ever get bored with using the BASIC command CLS to get rid of unwanted screen text? SCROLL, a listing included in this article, is an assembly language subroutine that can be called from a BASIC program when you want to clear the screen with a little added finesse. Unlike CLS, which erases the screen in one fell swoop, SCROLL removes the text more slowly. When the subroutine is called, the information slowly disappears by being moved into the center or off the top of the display. A programmer who writes in BASIC can use the routine to add variety to his programs.

We will also be taking a detailed look at some of the 8088's instructions for those who wish to learn more about assembly language.

Listing 1 is an example of how SCROLL is used in a BASIC program. To see how SCROLL works, simply type the program in while under BASIC's control. Of course, once the program is typed in, first save it to disk using the name "SCROLL.BAS" and then run it by pressing the F2 function key. Whatever you do, don't let all of those data statements discourage you. Let your PC do the work for you! Use BASIC's KEY statement to reassign function keys 7 and 8 (or any function keys you choose) to the strings ", &H" and "Data &H". By doing this reassignment, your typing time will be greatly reduced. (See your IBM BASIC text for details on how to reassign function keys in BASIC.) To use SCROLL in your own programs, just copy the data statements and load the hex values into memory with a READ statement in a for/next loop just as we have done.

The balance of this article is for those who want to learn some of the techniques used in 8088 assembly programming. The SCROLL procedures were written, not with the intention of showing how to write the most compact, nor the most efficient code, but rather to illustrate various programming techniques. (As a matter of fact, one short-cut that comes to mind immediately is to eliminate several PUSH and POP instructions. But, as in all good tutorials, I leave you with the challenge of learning how to do it.) It is more important to grasp the logic of the routine and to learn the principles than to develop shortcuts. After all, the Olympian trains before he competes.

SCROLL was chosen for its graphic orientation. As the program progresses you will actually be able to "watch" the execution of the code. You should make a visual correlation between the program and your screen. Those that have seen this routine have enjoyed it. It is fun.

As you study SCROLL, you will learn many concepts. The assumption is made that you know how to develop source code, how to use an assembler and how to link programs. What we will be concerned with are several instructions, among them are CLD, STD, MOVWS, and REP. Although these are covered in other texts, I have not seen them used in anything but the simplest routines. There will also be other techniques covered. We will look at a method for turning the cursor on and off, for avoiding "static" on the color monitor as we directly access the color video RAM when it is in text mode, for knowing if a machine has a color graphics card or a monochrome card installed, for linking a machine language subroutine to a BASIC program, and for passing a variable to the subroutine. Finally, we will write a simple procedure that shows how a routine can be written that will make it possible for the ROM interrupts to be accessed from BASIC.

Contrary to what you may have heard, assembly language programming is not difficult. It does take time to learn and to program in assembly, but the concepts are not hard to master. Most of the difficulty comes from the texts that have been written on the subject. Nearly everyone that wants to write programs in assembly code has some understanding of BASIC. This knowledge was, more often than not, gained from BASIC

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

programming texts. These texts are mostly written in a tutorial fashion that make the BASIC language easy to learn. This is not the case when it comes to texts on assembly. For the most part, these texts present concepts in a random manner making them difficult to learn. While they make good reference books, they make poor teaching guides. (This trend does seem to be changing as authors become more familiar with the PC. My suggestion to those who are looking for an assembly language text is to select one with a recent publication date. Such a date indicates that there was more time available for the author to write comprehensive explanations. And therefore, with all other things being equal, they should provide the best instruction.) This article will present the above concepts in more of a tutorial fashion which will make them easier to grasp. As we go along, some general ideas and rules will be presented that can save countless programming hours. Such ideas are common sense, but they are brushed over, or worse, they are omitted in most books.

Before we look at the details of the assembly language routine, let's take a few moments to review the aspects of the BASIC program that set up the call to SCROLL. There are several ways available to the programmer to load and store the machine language subroutine. These are discussed in appendix C of IBM's BASIC manual. We will look at the method which uses BASIC's CALL statement to branch to the routine. BASIC's READ, POKE, and DATA statements are used to load the routine into memory. The advantage to this method is that you do not have to keep a separate file that contains the subroutine on your disk since every time the program is loaded and run, the routine is poked into memory. The first thing we must do when writing a BASIC program that CALLs a machine language subroutine is to define the area in memory that will be used to store the routine. This is accomplished with the DEF SEG statement. For our purposes, we will assume that the BASIC program is large, and that it will be used on machines that have at least 96K of memory so we will reserve the upper 4K bytes of the 96K RAM area for our routine. See listing 1, line 20. (Nearly all PC's have a least 96K of RAM memory, so it seems foolish to take up part of BASIC's 64K of work space with an assembly language subroutine when it can be loaded just as well outside of BASIC's work area.)

After the memory has been reserved for the subroutine, the next step is to put the subroutine into memory. This is accomplished with a loop that reads the data statements and pokes the hex values into memory. Once this is completed, the program continues its execution and the subroutine is CALLed by giving the routine a name and by assigning a starting value to it. With SCROLL there is also a parameter that is passed. The parameter is given the name SPDD and this variable determines the rate of scroll. The higher the value, the slower the scroll. Since SPDD is an integer, the upper limit of its value is 32,767. The program does not allow for a value less than 1,750. The only reason for this is that if the screen scrolls too quickly, then you will not be able to observe the results! (If you want to see for yourself, go ahead and try it by setting the SPDD variable in line 110 to say 100. And while you are experimenting, why not try assigning a negative value to SPDD?)

Now that you know what the assembly language subroutine does, and how it can be called from BASIC, let us look at the details of how it was written and how it was translated into all of those data statements.

Listing 2 is the original assembly language source routine as it was created by the assembler and written to the file that has the LST extension. It goes beyond the scope of this article to take you through the step by step procedures for creating a source file. We will, however, concentrate on some of the principles of SCROLL that are not generally mentioned in the assembly language texts.

First, we must set up the routine to accept the parameter that is being passed to it by the calling routine; in our case, the BASIC program is that routine. Since BASIC does not save any of the values in the segment registers or the pointer or index registers, we must save the values with the appropriate PUSH instructions. The first register that must be PUSHed is the BP (base pointer) since its contents are changed by initializing it to mark the top of the stack. After the value of SP is moved to BP, the remaining registers need to be saved by pushing them onto the stack. (One note of caution: the stack that is created by BASIC's CALL, has for its default room to store only 16 bytes.) After this is accomplished, we fetch the address of the variable that is passed to the subroutine. In our case, since there is one variable, we know that its address will be found at BP + 6 bytes. (The address of the variable which we are trying to reach is the first byte pushed onto the stack by BASIC. The calling routine's segment address and offset address occupy the next four bytes and the first instruction of the assembler subroutine, PUSH BP, takes the last two bytes.) Since the assembler will not move the contents of the memory location pointed to by BP directly into a general purpose register, we must first move the value into SI and then move the contents of SI into a general purpose register. In this case, that register is BX. Now that we have recovered the value of BASIC's SPDD variable, we can proceed with the set-up for the routine.

Even though the value of the SPDD variable has been recovered from the BASIC program, we need to do several more things before the actual scroll takes place. Since the video RAM is located at different addresses for each of the two video cards, we must find out which of the two cards is attached to the PC. Line 23 CALLs the subroutine VIDCARD which decides which card is attached. Once it is established which card is mounted in the PC, the ES and DS segments are initialized to the starting address of that card. The result of the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

VIDCARD subroutine is passed back to the main procedure in the AX register.

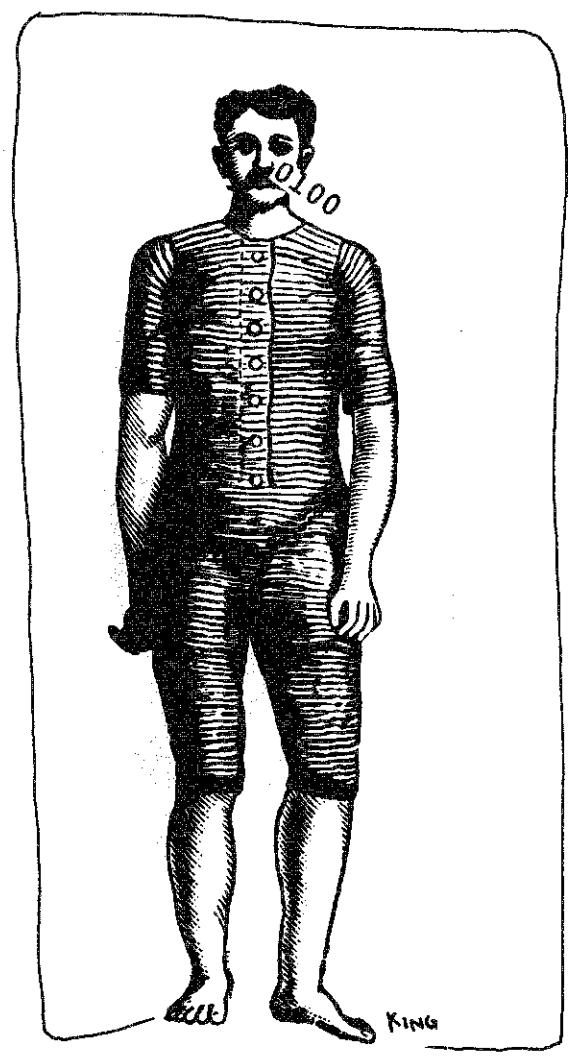
One final step remains. For aesthetic reasons the cursor needs to be turned off. Lines 24-28 send us to the proper routine depending upon whether the color or monochrome monitor is installed. The only difference in these two routines is the port address for the control of the cursor. The program could also have been written to establish which card is installed and then immediately branch and turn off the cursor. However, since the overall program is quite small and overhead is not a problem, many subroutines that accomplish only one task have been written in order to make it easier to follow the program. (If one wants to write a program for only a color monitor, then only the procedures that apply to the color card need to be copied from this program. This saves the complication of deleting lines from within a procedure.)

The preparation for the actual scroll starts at line 32. The CX (count) register is loaded with the number of characters to scroll. Since the first part of the routine scrolls the text to the center of the screen there are 40 positions to scroll in either direction for an 80 column display. The main routine has one primary loop labeled as LOOP1 and two subloops labeled as LOOP2 and LOOP3. LOOP2 moves each character on the right half of the screen one position to the left. LOOP3 moves each character on the left side of the screen one position to the right. These two loops are repeated 40 times by the primary loop, i.e., LOOP1.

We will look at LOOP2 in detail to understand some of the subtleties of the SCROLL program. Understand that LOOP3 is a mirror image of LOOP2. As we move into the setup for LOOP2 we first compare the contents of AL to 30H.

If the values are equal, the monochrome card is installed, and the routine jumps directly to SETUP1. However, if the monochrome card is not installed, then a branch to a procedure called VIDOFF occurs. The reason for VIDOFF is quite simple. Almost all users of the PC have run programs that generated static interference on the video screen. (An example of such a program can be found in the popular Norton Utilities. Just run UnErase on a color monitor and you will see what I mean.) This interference comes about when a program accesses the video RAM on the color graphics card. Unfortunately, though this problem creeps up in many programs (particularly those in the public domain), I have been able to find nothing written on the topic. (I only recently learned that the July 1983 issue of Softalk has a discussion on this subject. However, I am waiting for a back ordered issue to arrive and I cannot comment further on it.) I was able to discover one solution to this annoying problem only by reading the source listings for the BIOS which can be found in appendix A of the IBM Technical Reference Manual! (While not exactly exciting reading, it can be very educational. It's too bad that other computer manufacturers do not demonstrate the same attitudes that IBM does by publishing their ROM routines. Who says IBM does not support the needs of their customers!)

The purpose of the VIDOFF procedure is to solve the static problem by turning the video display off for a very brief moment during the the period that the routine is moving the strings in the video RAM. Since the display is off, there is no static on the monitor. This brief interruption of the image to the monitor is apparently one of the reasons why the PC does not have smooth scrolling. (While we are on the subject, another cause for the lack of smooth scrolling on our favorite micro is because scrolling is achieved by moving lines from one row to the row immediately above or beneath it in one



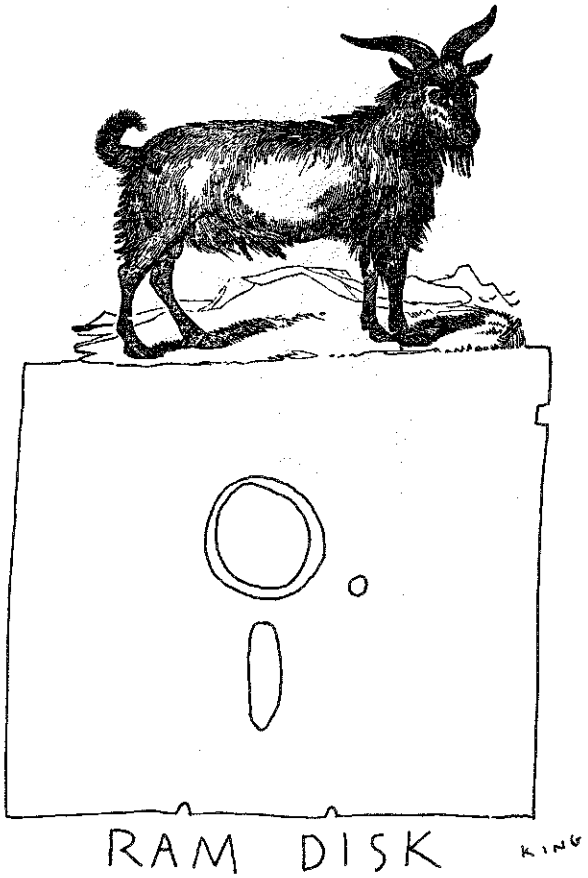
Famous Computer Personalities #89
Bernhard Castlesheep, the only documented case of a programmer actually to suffer from a code in the nose.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

step. The computers that achieve smooth scrolling do so by moving each line in small increments in the desired direction.)

One of the reasons why I choose to illustrate these points with SCROLL is because one can really get an appreciation of how assembly language is both flexible and fast in handling screen displays. For all of the time that the balance of the instructions in the loop are carried out, the video image is off if a color card is installed. In other words, 975 characters are each moved one position and a blank character is written to the end of the line, all in a little more time than it takes for your PC to blink its cursor!

With the video display set for the particular card installed in the PC, we now are prepared to set the value of the pointers and the direction of the string movement. The SI (source index) is initialized to 80, which is the address of the 40th character position in the video RAM. (There are two bytes used for each character in the video RAM. The even addresses contain the ASCII code of the character and the odd addresses contain the display attributes of the characters.) The DI (destination index) is initialized to 78, which is the 39th character position. The CLD instruction at line 39 clears the direction flag, which prepares for the MOVSW instruction at line 43. A CLD instruction makes the SI and DI registers increment two bytes after every MOVSW instruction is executed.



We finally have arrived at the heart of the program! Once the SI and DI registers are initialized, their values are pushed onto the stack. The CX register is then initialized with 39, which is the number of characters we need to move. The result of the MOVSW instruction is that the character at the address pointed to by the SI register is copied to the address pointed to by the DI register. So the character at the 40th position is moved to the 39th position in the video RAM. Then the indexes are automatically incremented by two bytes (again, one byte for the character and one byte for its attribute). The character at the 41st position is then copied to the 40th position. The REP instruction causes the MOVSW operand to execute 40 times (i.e., until CX reaches 0). Thus, the scroll is achieved. Once all forty characters are copied one position to the left (or more technically, moved one position down in the video RAM memory), a blank character is written to the end of the line.

After the blank character is written to the end of the line, the SI and DI registers are restored to their original value. The number 160 is added to each of their values which effectively sets them equal to the address in the video RAM that points to the center of the next line. DI is compared to the value of the center position of the 25th row. If it is equal to or less than that value (3918) then the program loops back to line 40 and another row of characters is scrolled one position to the left. If after the ADD instruction the value of DI is larger than 3918 we know that the program has moved the characters in all 25 rows. A check is performed to see if the color card is installed and if it is, the VIDON procedure is called to turn back on the video.

If it were not for the call to the next procedure, all of the above would be for naught. Assembly language is so fast that it requires a moment's delay to make the routine visually effective. The DLAY procedure accomplishes that function. A simple loop is executed to slow the PC down. The length of time that it takes to negotiate the loop depends upon the value in BX, which in turn is what was passed to the SCROLL subroutine by the BASIC program. (This is another example of the power of assembler. If the loop is not executed at least 1751 times then the delay is not long enough to allow the user to witness the effects of the scroll!)

After the delay, the entire procedure is repeated, but in reverse. LOOP3 accomplishes this for us. The only difference between LOOP2 and LOOP3 is the STD instruction on line 64 and the initial values of SI and DI. The STD (set direction flag) instruction causes the addresses in SI and DI to be decremented by two bytes when the MOVSW instruction is executed. As with LOOP2, LOOP3 is executed until all characters in every row are shifted one position. Again the DLAY procedure is called so that the effect has time to register upon the user. Then CX is restored so that the program can loop

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

back to LOOP1. LOOP1 is what is responsible for the program moving each character position 40 times.

When LOOP1 has been completed, the program falls through to the ending instructions. The cursor is moved to row one, column one and then it is turned on by the appropriate procedure. Each of the registers that were saved at the onset are restored, and a far return is used to send the program back to the calling BASIC program. The operand 2 causes the stack to be reset to the correct value. What is happening is that the parameter that was passed to the subroutine is cast off. The operand is always two times the number of parameters that are passed. This is the final instruction for the SCRLLCNT procedure.

The other procedure, SCRLLUP, is much simpler. (Thank goodness!) It is included here so that we can look at how to write a routine that gives BASIC access to the interrupts that are normally called through assembly language. As with the first procedure, the SPDD variable is passed to the subroutine and the registers are pushed onto the stack. The interrupt that we are concerned with here is number 10H. (Note: that is 10 hex, not 10 decimal) Interrupt 10H provides the programmer with direct access to the system's scroll routine. If we did not want to slow the routine down to watch what happens, it would be a simple matter to do the scroll. However, we do want to see the scroll, so we have to add several lines of code.

Line 110 initializes CX with the number of lines on the screen. We do this so that as the loop executes, we can call the DLAY procedure. To prepare for the interrupt call, DX is initialized with the row and column of the lower right corner of the display and CX is initialized with the row and column of the upper left corner. (It takes less time to perform XOR CX,CX than MOV CX,0.) By initializing CX or DX with different values, a program can be written to scroll only the upper part or the lower part of the display. Thus, the programmer can set up a window. The instruction at line 114 tells the interrupt routine to scroll up one line at a time. (MOV AX,0700 would make the routine scroll the entire screen down which would not allow for a pause.) And finally, the MOV BH,07 causes the blanked line to display the normal video attribute.

After the return from the interrupt, the entire screen will have been scrolled up one line. BX is then POPed to get the value of the SPDD variable and then the DLAY procedure is called. Upon return, the loop counter is POPed and the loop is executed. After the loop executes 25 times, the entire screen will have been cleared. The registers are restored to their initial value and a far return is performed.

Now that the routines are written, how can they be translated into the hex numbers that are required for the BASIC's data statements? Most of you with any assembly programming experience will be familiar with the LST file that is created by the assembler. (See listing 2 for an example.) The numbers on the extreme left of the printout are the line numbers as generated by EDLIN. The next set of four hex numbers are the offset addresses into the code segment. The balance of the hex numbers represents the translation of the 8088 instructions into machine language.

This listing, of course, is invaluable when one is writing code, for it provides a quick access to the program complete with comments and references. However, the listing is not complete enough for our purposes. When the LST file is created, it is not known where the subroutines (near procedures) are going to be loaded into memory. Thus, the lines that have code for CALLs are followed by an "R" and the offsets are subject to alteration by the linking process.

What we then need is the hex code for the actual EXE file. How do we go about getting this? We have all experienced the graphic characters that result from TYPE(ing) an EXE file. Obviously, this is not the solution to our problem. This is where DEBUG comes into play. DEBUG has a command that allows us to capture all of the translated source code after the linking step. The steps to do this are relatively simple. First, set your printer on line. Next run your EXE program under debug by typing in DEBUG SCROLL.EXE. When this is done you will see DEBUG's familiar dash prompt. Using the "R" command, look at the way the registers are initialized, taking note of the value in the CS register. We are now ready to dump the EXE file to the printer. While holding down the control key, press the grey PrtSc key. This will cause, of course, any subsequent displays to the monitor to be printed by your printer. On the command line (i.e., at the dash

Moving?

Please send us both your old and new addresses, with Zip Codes. If possible, include the address label from a recent issue of *Programmer's Journal.* Allow three to four weeks for processing.

Programmer's Journal
P.O. Box 30160
Eugene, OR 97403

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

prompt) type "U" (without quotes) for unassemble. Then type the segment address of the code segment which we made note of after we issued the "R" command. In this case the code segment is at 6316. (Your segment address probably will differ.) Since we want to dump to the printer the entire code for the EXE file we start at offset address "0000" and continue to offset 014A. (Look at the assembler's LST file output to find the format for entering the address on the command line.) We do not have to guess where the ending address is for our code. The LST file provides this information for us! We want to do the unassemble up to the last instruction that generates object code. In this particular case, that instruction is a RET which shows us that it is located at an offset 014AH bytes from the start of the code segment. (The ENDP, ENDS and END instructions are pseudo-ops which do not generate object code, and thus are of no concern to us.) Now press return, and the entire EXE file will be printed out showing a complete hex translation of the source code with all of the correct offsets. When the complete file is printed hit the control key along with the PrtSc key again to keep what is being typed from being echoed to the printer. Working with this printout will have one other advantage, as we shall soon see.

Even If You Don't Have The 8087...

debug 8

from RollinSoft

➲Extends & enhances the power of DEBUG by adding complete support for the 8087 numeric coprocessor.
➲Register values are displayed with 18-digit floating-point accuracy.
➲Status, Tag, and Control words are identified.

PLUS!

➲Enter decimal values directly into any 8087 register. Initialize the 8087 with one command.
➲Requires DOS 2.x DEBUG.

New Commands

K clear screen, set screen mode, attribute, and border
B binary and decimal conversions
H Hexarithmetic with 6 operations
Y one-step trace of CALL, INT, LOOP and REP. One-line trace display.
X 8087 npX display and modify
* timer-interrupt, FWAIT bugs fixed

Rush $49.95 to:
RollinSoft PO Box 12147, La Crescenta CA 91214
Created by Dan Rollins

Now we are ready to write the data statements that we need in our BASIC program. As already mentioned, you can save yourself a lot of typing by using BASIC's feature that allows reassigning of any of the function keys to be equal to any string. Once this is done, you are ready to type the hex numbers as printed out by DEBUG into BASIC data statements. Look at the BASIC program in listing 1 to see how the data statements are generated. Simply type in the hex numbers immediately following the offset address found in the EXE listing as given to us by DEBUG's unassemble command. The secret here is not to transpose any numbers! Although the lines with the data statement can contain any quantity of hex numbers, I choose to limit the length of the line to 10 hex numbers because it is easier to calculate the number of bytes used by the subroutine. We need to know the total number of bytes to set the upper limit of the for/next loop that pokes the hex values into memory.

Besides supplying the necessary number of bytes that the CALLed procedure is from the CALLing instruction, the unassemble command makes one other thing easier for us. As you undoubtedly know by now, the 8088 stores 2-byte addresses in memory in reverse order. When you look at the hex representation of the machine language of a program as it is displayed by the LST file, the addresses have been "corrected"; that is, the hex numbers are not printed in reverse order. Since we want to load into memory the hex representation of the program, we would have to watch for such occurrences of the bytes being "unreversed" and then reverse them again in our data statements. This in itself can cause errors in the data statements or at the least it will slow down their creation since watching for 4-byte addresses is just one more tedious detail. As you can see though, the listing generated by DEBUG's "U" command does not make any such "correction." And it becomes a much simpler matter to type in the hex numbers. Such simplicity itself makes it worth the effort of using DEBUG's listing even when the assembly language routine does not make any calls to near procedures.

So there we have it. We have written two procedures that will give our BASIC programs a little finesse, and we have learned some assembly language techniques for string handling and text displays. Now, see if you can apply what you have learned. If you were reading closely, you already know how to make the display scroll down. Try now to adapt the appropriate procedure to cause the display to scroll off to the right or left. Furthermore, you need only make a few simple alterations and the routine will scroll the lines in opposite directions. If you really want to get bold, write a routine using the principles illustrated to clear the screen by scrolling the text up and down and left and right at the same time so that the text will seem to disappear into a point in the center of the screen. Just like learning BASIC, the trial and error method will teach you a lot. So practice and explore!

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
1' ******** SCROLL.BAS ***********
10 KEY OFF
20 DEF SEG=&H1700 : REM SEGMENT ADDRESS OF LAST 4000 BYTES OF 96K RAM SPACE
30 FOR I = 0 TO 330 : REM 331 BYTE SUBROUTINE TO BE LOADED INTO MEMORY
40 READ J
50 POKE I,J
60 NEXT
70 FOR K = 0 TO 24
80 PRINT "The program is designed to CLEAR THE SCREEN
          by SCROLLING writting to the center"

90 NEXT
100 SCROLL = 0 : SPDD% = 5000 : REM SPDD% VARIABLE CONTROLS SPEED OF SCROLL
110 IF SPDD% < 1750 THEN SPDD% = 1750
120 CALL SCROLL(SPDD%)
130 REM STILL MUST MOVE CURSOR TO ROW/COL 0,0 BECAUSE BASIC SAVES POSITION OF
140 ' CURSOR BEFORE ASSEMBLY LANGUAGE SUBROUTINE CALL!
150 CLS
160 FOR K = 0 TO 24
170 PRINT "The program is designed to CLEAR THE SCREEN
          by SCROLLING writting up the screen"

180 NEXT
190 REM SCROLL SET TO 156 BECAUSE SCROLL UP ROUTINE STARTS 157 BYTES
200 ' FROM START OF ROUTINE
210 SCROLL = 156 : SPDD% = 30000
220 IF SPDD% < 1750 THEN SPDD% = 1750
230 CALL SCROLL(SPDD%)
240 CLS
250 END
260 DATA &H55,&H8B,&HEC,&H1E,&H06,&H57,&H56,&H8B,&H76,&H06
270 DATA &H8B,&H1C,&HE8,&HD7,&H00,&H3C,&H30,&H75,&H06,&HE8
280 DATA &H0F,&H01,&HEB,&H04,&H90,&HE8,&HE3,&H00,&HB9,&H28
290 DATA &H00,&H51,&H3C,&H30,&H74,&H03,&HE8,&HA1,&H00,&HBE
300 DATA &H50,&H00,&HBF,&H4E,&H00,&HFC,&H56,&H57,&HB9,&H27
310 DATA &H00,&HF3,&HA5,&HB0,&H20,&HAA,&H5F,&H5E,&H81,&HC6
320 DATA &HA0,&H00,&H81,&HC7,&HA0,&H00,&H81,&HFF,&H4E,&H0F
330 DATA &H76,&HE6,&H3C,&H30,&H74,&H03,&HE8,&H89,&H00,&HE8
340 DATA &H8F,&H00,&H3C,&H30,&H74,&H03,&HE8,&H6F,&H00,&HBE
350 DATA &H4C,&H00,&HBF,&H4E,&H00,&HFD,&H56,&H57,&HB9,&H27
360 DATA &H00,&HF3,&HA5,&HB0,&H20,&HAA,&H5F,&H5E,&H81,&HC6
370 DATA &HA0,&H00,&H81,&HC7,&HA0,&H00,&H81,&HFF,&H4E,&H0F
380 DATA &H76,&HE6,&H3C,&H30,&H74,&H03,&HE8,&H57,&H00,&HE8
390 DATA &H5D,&H00,&H59,&HE2,&H98,&H3C,&H30,&H75,&H06,&HE8
400 DATA &HA6,&H00,&HEB,&H04,&H90,&HE8,&H7A,&H00,&H5E,&H5F
410 DATA &H07,&H1F,&H5D,&HCA,&H02,&H00,&H55,&H8B,&HEC,&H8B
420 DATA &H76,&H06,&H8B,&H1C,&H1E,&H06,&H57,&H56,&HB9,&H19
430 DATA &H00,&H51,&H53,&HBA,&H4F,&H18,&HB8,&H01,&H06,&HB7
440 DATA &H07,&H33,&HC9,&HCD,&H10,&H5B,&HE8,&H24,&H00,&H59
450 DATA &HE2,&HEB,&H5E,&H5F,&H07,&H1F,&H5D,&HCA,&H02,&H00
460 DATA &HBA,&HDA,&H03,&H50,&HEC,&HA8,&H08,&H74,&HFB,&HB0
470 DATA &H25,&HB2,&HD8,&HEE,&H58,&HC3,&H50,&HB0,&H0D,&HBA
480 DATA &HD8,&H03,&HEE,&H58,&HC3,&H8B,&HCB,&HE2,&HFE,&HC3
490 DATA &HCD,&H11,&H25,&H30,&H00,&H50,&H3C,&H30,&H75,&H06
500 DATA &HB8,&H00,&HB0,&HEB,&H04,&H90,&HB8,&H00,&HB8,&H8E
510 DATA &HC0,&H8E,&HD8,&H58,&HC3,&H50,&HBA,&HD4,&H03,&HB0
520 DATA &H0A,&HEE,&HBA,&HD5,&H03,&HB0,&H20,&HEE,&H58,&HC3
530 DATA &H50,&HB4,&H02,&HB7,&H00,&H33,&HD2,&HCD,&H10,&HBA
540 DATA &HD4,&H03,&HB0,&H0A,&HEE,&HBA,&H5D,&H03,&HB0,&H0B
550 DATA &HEE,&H58,&HC3,&H50,&HBA,&HB4,&H03,&HB0,&H0A,&HEE
560 DATA &HBA,&HB5,&H03,&HB0,&H20,&HEE,&H58,&HC3,&H50,&HB4
570 DATA &H02,&HB7,&H00,&H33,&HD2,&HCD,&H10,&HBA,&HB4,&H03
580 DATA &HB0,&H0A,&HEE,&HBA,&HB5,&H03,&HB0,&H0B,&HEE,&H58
590 DATA &HC3
```

```
                PAGE 60,132


;
TITLE   SCROLL.ASM SCREEN: Routine to scroll screen to clear
;THIS PROGRAM SCROLLS THE ENTIRE SCREEN TO CLEAR
;THEN IT HOMES THE CURSOR AND PROMPT
;
;
CODE    SEGMENT PARA 'CODE'
SCRLLCNT PROC   FAR
        ASSUME  CS:CODE
        PUBLIC  SCRLLCNT
;
        PUSH    BP              ;SAVE BP TO SET UP AS POINTER
        MOV     BP,SP           ;SET BASE PARM LIST
        PUSH    DS              ;SAVE ALL SEGMENT AND POINTER
        PUSH    ES              ;REGISTERS SINCE THEY ARE ALTERED
        PUSH    DI              ;BY THE PROGRAM
        PUSH    SI
        MOV     SI,[BP]+6       ;GET VALUE OF SPEED PARAMETER
        MOV     BX,[SI]         ;AND MOVE IT TO BX
;
;DETERMINE TYPE OF MONITOR CARD INSTALLED AND STORE IN AX REGISTER
        CALL    VIDCARD
        CMP     AL,30H          ;00110000B, IS IT MONOCHROME CARD?
        JNE     COLOR1          ;IF NOT, THEN FOLLOW COLOR ROUTINE
        CALL    BWCUROF         ;BLACK & WHITE CURSOR OFF
        JMP     COUNT
COLOR1: CALL    CCUROFF         ;COLOR CURSOR OFF
;
;START SCROLLING PROCEDURE
;
COUNT:  MOV     CX,40   ;CHARACTERS TO SCROLL FOR 80 COL. DISPLAY
LOOP1:  PUSH    CX
        CMP     AL,30H          ;CHECK FOR MONOCHROME CARD
        JE      SETUP1
        CALL    VIDOFF          ;IF COLOR CARD VIDEO MUST BE OFF
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
SETUP1: MOV     SI,80           ;POSITIOR RIGHT OF CENTER, 1ST ROW
        MOV     DI,78           ;ONE CHARACTER TO THE LEFT
        CLD
LOOP2:  PUSH    SI
        PUSH    DI
        MOV     CX,39           ;CHARACTERS TO MOVE, LAST POSITION BLANK
        REP     MOVSW           ;MOVE EVERY CHARACTER ONE POSITION
        MOV     AL,' '          ;AT END WRITE BLANK TO DISPLAY,TOTAL 40
        STOSB
        POP     DI
        POP     SI
        ADD     SI,160          ;MOVE TO NEXT LINE
        ADD     DI,160
        CMP     DI,3918         ;CENTER POSITION OF 25TH ROW
        JBE     LOOP2           ;IF GREATER THAN BOTTOM CENTER, QUIT
        CMP     AL,30H          ;IF B&W CARD
        JE      STALL           ;THEN GO STRAIGHT TO DELAY LOOP
        CALL    VIDON           ;OTHERWISE TURN VIDEO ON AFTER RETRACE
;
STALL:  CALL    DLAY
;
;START SCROLL FROM LEFT EDGE TO CENTER
        CMP     AL,30H          ;CHECK TO SEE IF MONOCHROME CARD
        JE      SETUP2
        CALL    VIDOFF          ;REMEMBER VIDEO OFF IF COLOR CARD
SETUP2: MOV     SI,76           ;CENTER POSITON -1, 1ST ROW
        MOV     DI,78
        STD                     ;MAKES SI & DI DECREMENT IN MEMORY
LOOP3:  PUSH    SI
        PUSH    DI
        MOV     CX,39
        REP     MOVSW
        MOV     AL,' '
        STOSB
        POP     DI
        POP     SI
        ADD     SI,160
        ADD     DI,160
        CMP     DI,3918
        JBE     LOOP3
        CMP     AL,30H
        JE      STALL2
        CALL    VIDON
;
STALL2: CALL    DLAY
;
        POP     CX
        LOOP    LOOP1
;
;HOME CURSOR AND ENDING ROUTINE
        CMP     AL,30H
        JNE     COLOR2
        CALL    BWCURON         ;REMEMBER TURN CURSOR ON
        JMP     RESTORE
COLOR2: CALL    CCURON          ;COLOR CURSOR ON

RESTORE:POP     SI
        POP     DI
        POP     ES
        POP     DS
        POP     BP
        RET     2               ;DISCARD 2 BYTES FOR VARIABLE
SCRLLCNT ENDP
;
SCRLLUP PROC    FAR
        PUBLIC  SCRLLUP
        PUSH    BP
        MOV     BP,SP
        MOV     SI,[BP]+6       ;MOVE ADDRESS OF PASSED PARAMETER TO SI
        MOV     BX,[SI]         ;SAVE CONTENTS OF ADDRESS POINTED BY SI
        PUSH    DS
        PUSH    ES
        PUSH    DI
        PUSH    SI
        MOV     CX,25           ;NUMBER OF ROWS TO SCROLL UP
NEWROW: PUSH    CX
        PUSH    BX
        MOV     DX,184FH        ;ROW/COL. OF LOWER RIGHT CORNER
        MOV     AX,0601H        ;SCROLL UP ONE LINE
        MOV     BH,07           ;NORMAL VIDEO ATTRIBUTE (B&W)
        XOR     CX,CX           ;ROW/COL. OF UPPER LEFT CORNER (0000)
        INT     10H             ;BIOS INTERRUPT CALL
        POP     BX
        CALL    DLAY    ;DELAY ROUTINE TO SLOW SCROLLING OF EACH ROW
        POP     CX
        LOOP    NEWROW          ;25 LOOPS OR NUMBER OF ROWS TO SCROLL
        POP     SI
        POP     DI
        POP     ES
        POP     DS
        POP     BP
        RET     2               ;DISCARD 2 BYTES FOR VARIABLE
SCRLLUP ENDP
;
;
VIDOFF  PROC    NEAR            ;VIDEO OFF DURING VERTICAL RETRACE
        MOV     DX,3DAH         ;COLOR CARD
        PUSH    AX              ;SAVE TYPE OF MONITOR CARD IN AX
WAIT:   IN      AL,DX           ;GET PORT
        TEST    AL,8            ;00001000B, WAIT FOR RETRACE
        JZ      WAIT
        MOV     AL,25H          ;00100101B, BIT PATTERN TO DISABLE VIDEO
        MOV     DL,0D8H         ;DX=3D8
        OUT     DX,AL           ;TURNS OFF VIDEO DURING RETRACE
        POP     AX
        RET
VIDOFF  ENDP
;
VIDON   PROC    NEAR            ;TURNS VIDEO BACK ON
        PUSH    AX
        MOV     AL,1101B        ;BIT PATTERN TO ENABLE VIDEO
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        MOV     DX,03D8H        ;LOCATION OF COLOR CARD PORT
        OUT     DX,AL           ;SENDING PATTERN TO PORT ENABLES VIDEO
        POP     AX
        RET
VIDON   ENDP
;
DLAY    PROC    NEAR            ;SIMPLE DELAY LOOP
        MOV     CX,BX           ;BX CONTAINS VALUE PASSED TO PROGRAM
DLOOP:  LOOP    DLOOP
        RET
DLAY    ENDP
;
;
;*************************************************************************
;  THE PROCEDURE VIDCARD DETERMINES WHAT VIDEO CARD IS INSTALLED
;  AND INITIALIZE ES AND DS REGISTERS (SEGMENTS) WITH
;  STARTING ADDRESS OF THE VIDEO CARD THAT IS INSTALLED
;  MOVSW REQUIRES THAT THE SOURCE STRING BE IN THE DATA SEGMENT
;  AND THE DESTINATION STRING TO BE IN THE EXTRA SEGMENT
;  SINCE THE SEGMENTS ARE BOTH DEFINED AS EQUAL TO THE VIDEO RAM, WE
;  USE MOVSW TO COPY A STRING FROM ONE PART OF THE VIDEO DISPLAY TO
;  ANOTHER
;  AX INITIALIZED SO THAT IT CONTAINS BIT VALUES FOR CARD INSTALLED
;*************************************************************************
;
VIDCARD PROC    NEAR            ;WHAT VIDEO CARD?
        INT     11H             ;INTERRUPT FOR EQUIPMENT DETERMINATION
        AND     AX,00110000B    ;ALL BITS SET TO 0 UNLESS MONOCHROME
        PUSH    AX              ;CARD IS INSTALLED THEN 4 & 5 SET TO 1
        CMP     AL,30H          ;00110000B
        JNE     COLOR           ;NOT EQUAL, THEN COLOR CARD INSTALLED
        MOV     AX,0B000H       ;STARTING ADDRESS OF MONOCHROME CARD
        JMP     SET_SEG
COLOR:  MOV     AX,0B800H       ;STARTING ADDRESS OF COLOR CARD
SET_SEG:MOV     ES,AX           ;SET UP EXTRA SEGMENT WITH CARD ADDRESS
        MOV     DS,AX           ;SET UP DATA SEGMENT WITH CARD ADDRESS
        POP     AX
        RET
VIDCARD ENDP
;
CCUROFF PROC    NEAR            ;TURN CURSOR ON COLOR CARD OFF
        PUSH    AX
        MOV     DX,03D4H        ;ADDRESS REGISTER
        MOV     AL,10
        OUT     DX,AL
        MOV     DX,03D5H        ;DATA REGISTER
        MOV     AL,20H          ;CODE TO TURN CURSOR OFF, 00100000B
        OUT     DX,AL
        POP     AX
        RET
CCUROFF ENDP
;
; NOTE: SETTING CURSOR POSITION WILL NOT BE EFFECTIVE IF ROUTINE IS
;       BEING CALLED FROM BASIC.  BASIC MUST SAVE CURSOR POSITION
;       BEFORE BRANCHING TO ROUTINE
;       METHOD INCLUDED HERE IN CASE ROUTINE IS TO BE USED WITH
;       OTHER CALLING LANGUAGE
;
CCURON  PROC    NEAR            ;TURN CURSOR ON COLOR CARD ON
        PUSH    AX
        MOV     AH,02           ;FUNCTION TO SET CURSOR POSITION
        MOV     BH,00           ;PAGE NUMBER
        XOR     DX,DX           ;POSITION OF CURSOR, ROW 0 & COL. 0
        INT     10H             ;VIDEO INTERRUPT
        MOV     DX,3D4H         ;PORT USED TO SELECT ALL REGISTERS
        MOV     AL,10           ;00001010B, CURSOR START REGISTER
        OUT     DX,AL           ;CURSOR START REGISTER SELECTED
        MOV     DX,3D5H         ;ADDRESS OF DATA REGISTERS IN 6845 CHIP
        MOV     AL,11           ;00001011B, CURSOR ENDING REGISTER
        OUT     DX,AL           ;CURSOR NOW ON
        POP     AX
        RET
CCURON  ENDP
;
BWCUROF PROC    NEAR            ;CURSOR ON MONOCHROME CARD OFF
        PUSH    AX
        MOV     DX,3B4H         ;GET PORT
        MOV     AL,10
        OUT     DX,AL
        MOV     DX,3B5H         ;GET PORT
        MOV     AL,20H
        OUT     DX,AL
        POP     AX
        RET
BWCUROF ENDP
;
BWCURON PROC    NEAR            ;TURN B&W CURSOR ON
        PUSH    AX
        MOV     AH,02
        MOV     BH,00
        XOR     DX,DX
        INT     10H
        MOV     DX,3B4H         ;GET PORT
        MOV     AL,10
        OUT     DX,AL
        MOV     DX,3B5H         ;GET PORT
        MOV     AL,11
        OUT     DX,AL
        POP     AX
        RET
BWCURON ENDP
;
CODE    ENDS
        END     SCRLLCNT        ;NAME OF FIRST PROCEDURE IN SOURCE □
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Join the network

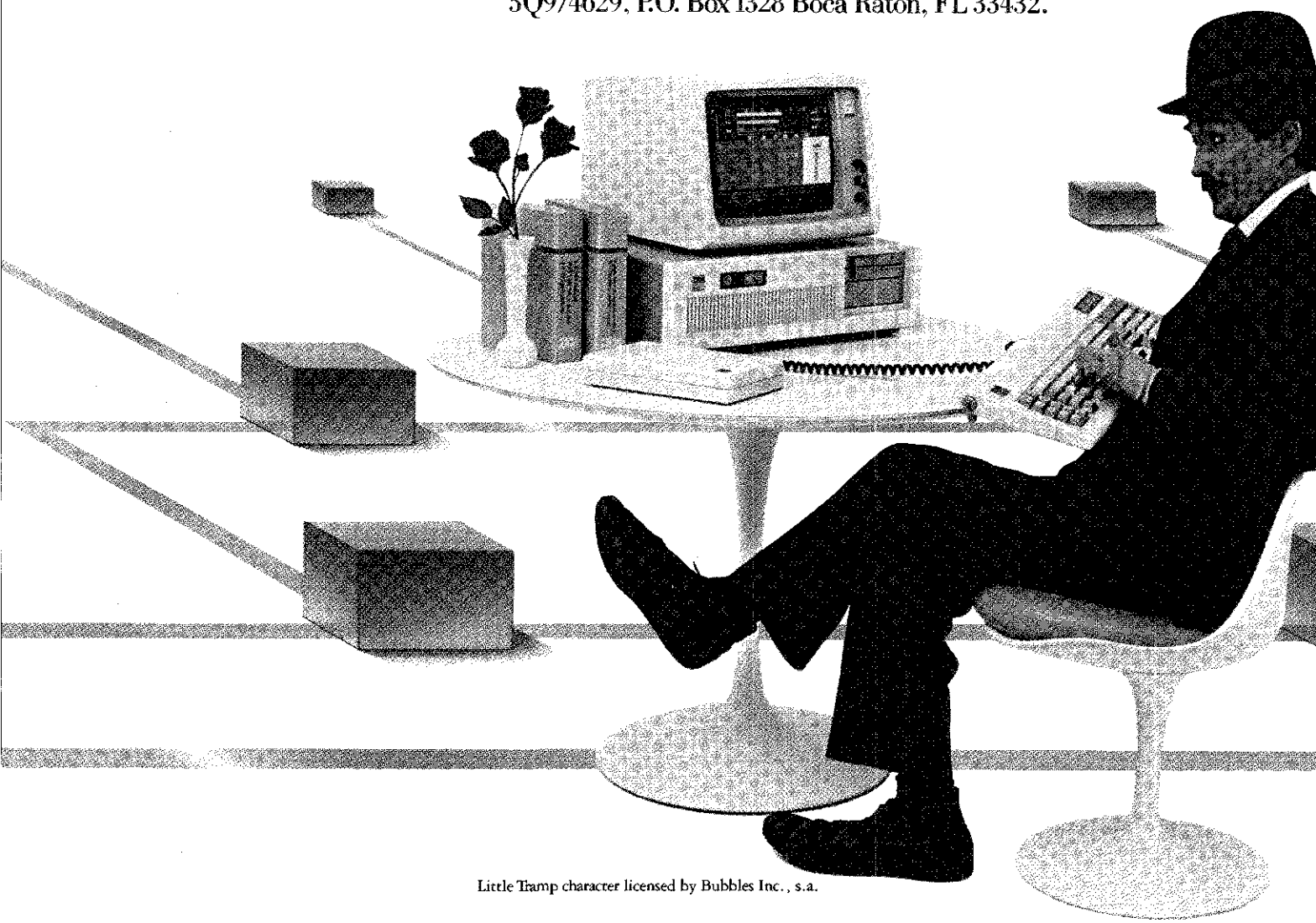
Develop a program for the new IBM PC Network.

Promote yourself into the top ranks of network programmers. Write new programs or modify existing ones to run on the local area network that's sure to be a best seller.

Capitalize on a chance to be in from the beginning with integrated business applications, productivity tools and office automation programs.

A high-level interface card makes programming for the IBM PC Network fast and easy. Full data sharing and byte locking capability, multiple servers, two megabits per second data transmission and a complete diagnostic package make the network equally as attractive to the end user.

For complete details about joining the network call 1-800-426-2700. Or write IBM Corporation, Editor, IBM Personal Computer Seminar Proceedings, 5Q9/4629, P.O. Box 1328 Boca Raton, FL 33432.



Little Tramp character licensed by Bubbles Inc., s.a.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

IBM PC Network Specifications

IBM PC NETWORK ADAPTER

HARDWARE HIGHLIGHTS

MICROPROCESSORS
- 6MHz 80188
- 6MHz 82586

MEMORY
- 32K PROTOCOL ROM
- 16K RAM
- 8K NET BIOS ROM

VIDEO COMPATIBLE RF MODEM
- TRANSMIT 50.75 MHz (CH T14)
- RECEIVE 219 MHz (CH J)
- SUPPORTS 1000 NODES
- MAXIMUM DISTANCE 5KM RADIUS FROM HEADEND
- MULTIPLE SERVICES POSSIBLE

DIAGNOSTICS
- POWER-ON SELF-TEST
- ON-LINE MEDIA MONITORING

OTHER
- SUPPORTS DMA DATA TRANSFERS
- 2-MEGABIT/SECOND DATA RATE
- MID-SPLIT BROADBAND

FIRMWARE HIGHLIGHTS

OPEN ARCHITECTURE
- PEER-TO-PEER NETWORK
- OPERATING SYSTEM INDEPENDENT
- LOCALNET/PC™, PUBLISHED LAYERED PROTOCOL

FUNCTIONS
- BASE FUNCTIONS PROCESSED ON THE ADAPTER, NOT THE PC
- DISTRIBUTED NAME SUPPORT
- REMOTE PROGRAM LOAD
- 32 CONCURRENT TWO-WAY SESSIONS
- HIGH THROUGHPUT RATE AT SESSION LAYER
- CHARACTER SET INDEPENDENT

IBM PC NETWORK CABLING SYSTEM

IBM PC NETWORK TRANSLATOR UNIT

HARDWARE HIGHLIGHTS
- SINGLE RF CHANNEL CONVERSION
- ATTACHMENT OF UP TO 72 PCs WITH IBM CABLING
- ATTACHMENT OF UP TO 256 PCs WITH CUSTOM CABLING
- DATA ONLY
- ALLOWS NODES WITHIN A 1000-FOOT RADIUS

CABLE

TYPE
- STANDARD CATV MEDIA (75 OHM COAX)
- TREE TOPOLOGY
- CATV STANDARD F-CONNECTORS
- PREBALANCED BROADBAND NETWORK

KITS
- BASE EXPANDER (ALLOWS EXPANSION FROM TRANSLATOR)
- SHORT-DISTANCE KIT (1 FOOT ADDITIONAL CABLE)
- MEDIUM-DISTANCE KIT (400 FEET ADDITIONAL CABLE REQUIRED)
- LONG-DISTANCE KIT (800 FEET ADDITIONAL CABLE REQUIRED)
- CABLE AVAILABLE IN 4 LENGTHS: 25 FT., 50 FT., 100 FT., AND 200 FT.

MAXIMUM PCs AND DISTANCES SUPPORT	RADIUS FROM TRANSLATOR	PCS
• TRANSLATOR ONLY	200 FEET	8
• 8 SHORT-DISTANCE KITS	200 FEET	72
• 8 MEDIUM-DISTANCE KITS	600 FEET	72
• 8 LONG-DISTANCE KITS	1,000 FEET	72
• 8-KIT COMBINATION	200 to 1,000 FEET	72

IBM PC NETWORK SOFTWARE

DOS 3.1

EXPANDED SUPPORT FOR NETWORKING
- FILE SHARING
- RECORD LOCKING DOWN TO BYTE LOCKING

PROGRAM INTERFACE TO NETWORK SOFTWARE
- REDIRECTION CONTROL
- INSTALLATION CHECKING
- DIRECT EXECUTION OF NET BIOS FUNCTIONS
- MULTIPLE SERVERS

IBM PC NETWORK PROGRAM

FULL SCREEN INTERFACE AVAILABLE

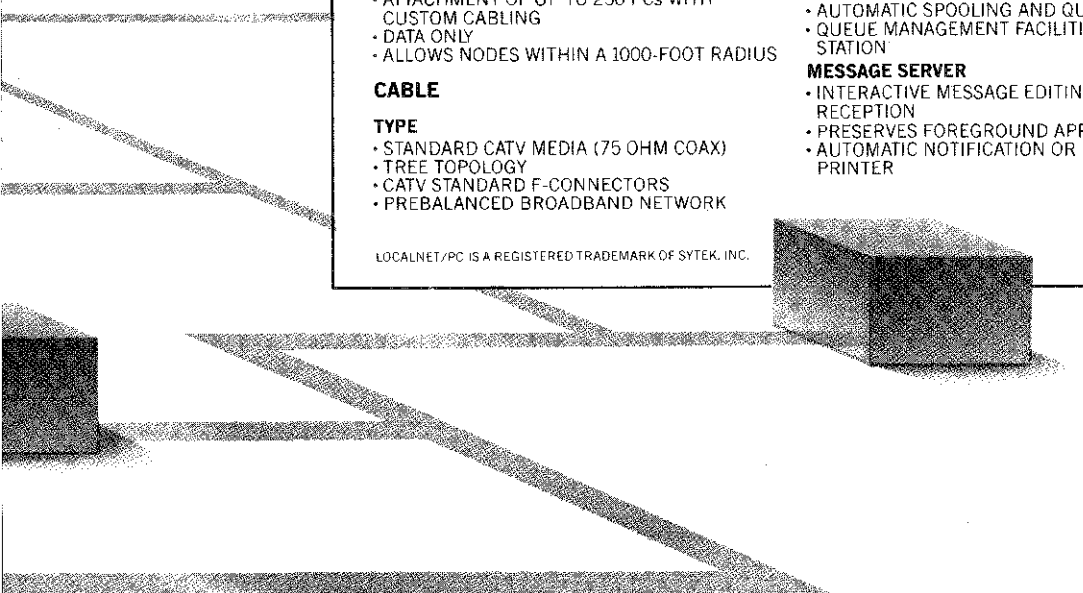
REDIRECTOR
- ALLOWS USE OF SHARED PRINTERS
- ALLOWS USE OF SHARED DISKS AND DIRECTORIES
- PROVIDES CAPABILITY TO SEND MESSAGES

FILE SERVERS
- SHARED USE OF NAMED DISKS OR SUBDIRECTORIES
- PASSWORD PROTECTION AGAINST UNAUTHORIZED ACCESS
- VARIETY OF ACCESS MODES SUPPORTED (E.G. READ ONLY)
- RECORD LOCKING TO CONTROL MULTIPLE UPDATES

PRINT SERVER
- SHARED USE OF NAMED PRINTERS
- PASSWORD PROTECTION AGAINST UNAUTHORIZED ACCESS
- AUTOMATIC SPOOLING AND QUEUING OF OUTPUT
- QUEUE MANAGEMENT FACILITIES ON SERVER STATION

MESSAGE SERVER
- INTERACTIVE MESSAGE EDITING/TRANSMISSION/RECEPTION
- PRESERVES FOREGROUND APPLICATION CONTEXT
- AUTOMATIC NOTIFICATION OR LOGGING TO DISK/PRINTER

LOCALNET/PC IS A REGISTERED TRADEMARK OF SYTEK, INC.



IBM

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PROFESSIONAL BASIC™

Access All The Memory Of Your PC!

Professional BASIC™ - the first of it's kind: a powerful programming language for the 16-bit microcomputer. You can simultaneously view program execution and see the change in variables, the code being executed, file buffers, or an "instant replay" of execution within the multi-window environment.

"The real magic of Professional BASIC™ is its wealth of 'windows' into an executing program." "I'm frankly amazed; My hat is off to Dr. Bennett...An elegant piece of coding indeed."
Personal Computer Age Dan Rollins, Feb. 1984

"Professional BASIC™...cranked out a timing and absolute error result competitive with the best compilers, while simply blowing the doors off every other known microcomputer interpreter."
Dr. Dobb's Journal Ray Duncan, Aug. 1984

FEATURES

- Interpreter language.
- Runs most IBM® PC BASIC programs.
- 8087 numeric coprocessor support.
- Have a 200x200 array (160k).
- Semi-compiles for immediate error checking and faster execution.
- Character-by-character syntax checking.
- More than 18 trace (half/full) windows.
- BCD arithmetic option, with 8087 (16 digit accuracy).
- Labeled lines. Use of 'GOTO label' statements.
- Save programs (or subroutines) without line numbers.
- Cross referencing of variables.
- 'SEARCH text' command.
- Large real numbers (10^{308} to 10^{-308}).
- Large integers (over 2 billion).
- Large files (over 4 billion records).
- Complete-sentence error messages.
- Runs on the IBM® PC and other compatibles.

PROFESSIONAL BASIC™
only $345
Version 1.08
new version of product

Morgan Computing Co.
10400 N. Central Expwy., Suite 210
Dallas, Texas 75231
(214) 739-5895

THE MOST EXTENSIVE LIBRARIES ANYWHERE
for the IBM PC, XT

THE GREENLEAF FUNCTIONS

Total Access to the PC

PROFESSIONAL PROGRAMMER'S TOOLS, With Comprehensive Manual, Full Source, Examples, Demos, & Newsletter.

Supports All Major Compilers, All Memory Models, DOS 1.1, 2.0, 2.1

IBM • Honeywell • Control Data • GE • Lotus . . .
Hospitals • Universities • Government • Aerospace . . .
Why not for your application?

Save Months of Painstaking Research

GENERAL LIBRARY – What You Would Write If You Had The Time
• 200 functions • DOS 2.0 • Video • String • Printer • Async • Color • Time & Date • Function Keys • Diagnostic • More

#L2 for Lattice or Microsoft, or #C2 for CI C86 . . . . . $175.00
#D1 for DeSmet C . . . . . $ 99.00

ASYNC COMM LIBRARY – Interrupt Driven Communications Capability
• Polled Mode • Interrupt Mode • Ring Bufferred • Modem Control • CRC • XMODEM Protocol Pieces • More

#L3 for Lattice or Microsoft, of #C2 for CI C86 . . . . . $160.00
#D3 for DeSmet C . . . . . $ 85.00

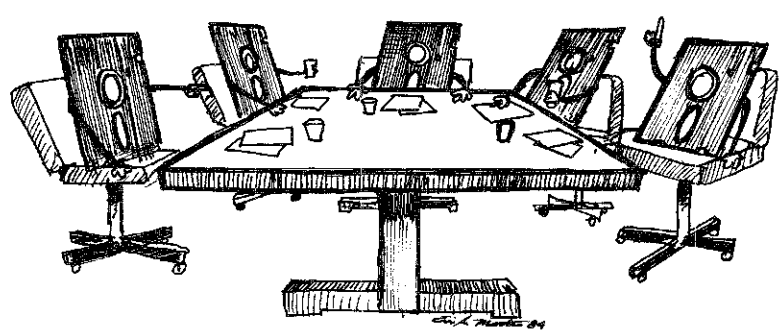
#CC1 Computer Innovations C86 Compiler V2.1 . . . . . $349.00
#CC2 Lattice C Compiler V2.1 . . . . . $395.00

GREENLEAF SOFTWARE ©

(214) 446-8641
• Order Direct or Ask Your Dealer
• MasterCard or VISA Accepted

• Add $7.00 Shipping Per Item
• Specify Compiler and Our Part Number

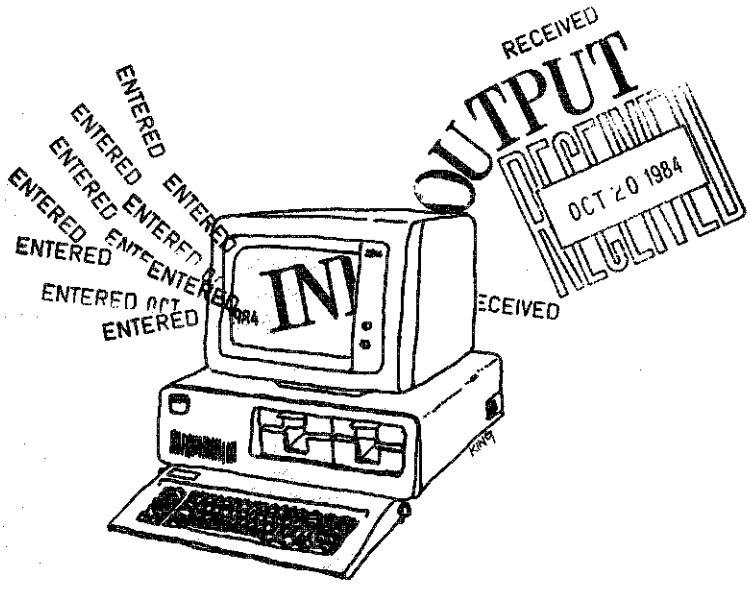
GREENLEAF SOFTWARE, INC.
2101 HICKORY DRIVE • CARROLLTON, TEXAS 75006



A DISKUSSION

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SCREEN I/O ROUTINES



by Joe Rayhawk and Stan Radzio

Copyright © 1984 by RAYHAWK AUTOMATION, NW, INC

This article is divided into two sections. The first discusses some of the technical issues involved in developing the Whizzard Screen I/O Routines. The second section gives a detailed description of how the individual routines work. You need to read this only if you are interested or if the first section is a little too concise.

MOTIVATION FOR THE EFFORT

I've seen a fair number of articles with titles such as: "What IBM Did Right/Wrong With the PC, PC/XT, PCjr". Several talk about the slow BIOS screen interface. I wanted to do something about those complaints. So, I rewrote the BIOS screen interface; a complete replacement for interrupt 10. I squeezed every bit of speed I could out of that code. I used faster instructions, table look-ups instead of multiply instructions, twists in the logic to avoid a divide instruction, and very tight code.

What did I get for my trouble? An 8% improvement in the TYPE command, an 11% improvement in BASICA, and a 10% improvement in WordStar. We're talking about something not worth the money here. There is a principle of physics involved, the Conservation of Difficulty. Given the need to support both the graphics and monochrome card, given the ability to segment the buffer on the graphics card into four or eight pages, there is a certain amount of work required to set up the registers for screen I/O, and that work is a conserved quantity.

Aha, you say! If I can't eliminate that work, what happens if I send a whole stream of characters to the screen at once instead of a single letter at a time?

This is a great deduction on your part, Sherlock. We can do the register setup once, then send out a whole stream of characters. As it is now, the BIOS screen interface will only print one character at a time on the screen. (BIOS will repeat a single character several times but this facility is not terribly useful except to blank out a line.) There is a DOS call (AH=9) which appears to handle a stream of characters. Actually, this call breaks up the string and sends it to BIOS one character at a time. Speaking of this DOS call, whose idea was it to use the dollar sign ($) as the string terminator? (Miss Marfle, please get that man on the phone and bring in my air horn.)

Besides sitting around complaining, we can respond by writing a subroutine to take the place of the missing BIOS call. Programs that do this, like the IBM Personal Editor, are called memory-mapped in some magazines. This makes for very snappy screens for programs that you write yourself. We have provided a set of routines that do this if you are willing to code the CALLs in your BASIC programs. They are included on our club diskette. They were the easy part. I also got very tired of the hassles of doing CALLs in the interpreter and created a technique to make it source-compatible with the compiler approach. This technique is also included.

Now, what can we do for programs we have bought off the shelf? With a little patience and a little luck, we can discover a critical area in many programs where the data to be displayed

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

on the screen is still tied together as a string. The program area will probably look like:

```
(type A)            MOV     CX,CHAR_COUNT
                    MOV     SI,STRING_ADDRESS
              LABEL:LODSB           ; load a character
                    CALL    OUTPUT_CHAR
                    LOOP    LABEL
```

The subroutine OUTPUT_CHAR will typically PUSH CX, move a 1 into CX (display one character per call), issue an INT 10, move the cursor over one to the right, POP CX, then return.

The other usual layout looks like:

```
(type B)            MOV     SI,STRING_ADDRESS
              LABEL:LODSB           ; load a character
                    OR      AL,AL
                    JZ      DONE
                    CALL    OUTPUT_CHAR
                    JMP     LABEL
              DONE:
```

This piece of code operates on a string where the length is unknown but the end of the string is tagged by a zero (null). Only someone with tapioca for brains would use a dollar sign as a terminator.

Once you find the critical piece of code within a program, you can modify the CALL OUTPUT_CHAR to call your own subroutine, or you can replace the call with an unused interrupt, like INT 68. Your video subroutine or interrupt handler can then grab up the whole string, send it to the screen in one shot, then return to the main program with conditions set to terminate the loop (leave a 1 in CX or leave SI pointing to a 0).

On CLUBware Software Diskette #1, there are three examples of this kind of program modification - one for the BASIC Interpreter, one for Compiled BASIC programs that use BASRUN.EXE, and one for compiled BASIC programs which do not use BASRUN.EXE. Now we leave theory behind and enter the very complicated real world. Real programs do not have just a type A structure or just a type B structure; most use both techniques. For instance, the BASIC Interpreter uses type A for PRINT statements writing text and type B for error messages. The BASIC compiler uses type A for printing character strings and type B for printing numbers.

Both the BASIC Interpreter and the compiler also have a type C structure which looks like:

```
(type C)            MOV     AL,0D
                    CALL    OUTPUT_CHAR
```

to be used when putting a carriage return on the end of a line. Both the compiler and the Interpreter use such a weird technique for keeping track of where carriage returns have been written and where they might be needed (and when the screen should scroll) that we don't want to mess with type C. This brings up the next point: if you replace a subroutine like OUTPUT_CHAR in someone else's program, keep a copy of the original address in your back pocket. If you get into a situation that your replacement subroutine can't handle, you can pass off the call to the original code.

OVERVIEWS

Here is how one of the CLUBware routines, BASPRINT, can be used to speed up interpreter PRINT statements:

1) Execute BASPRINT to make our code resident. (Do this once after booting the system.)

2) Bring up the BASIC Interpreter by invoking either: BASIC or BASICA.

3) When BASIC clears the screen, BASPRINT intercepts the I/O and takes this opportunity to point the interrupt vector for INT B4 to itself.

4) The OUTPUT_CHAR subroutine for BASIC is burned into system ROM so it can't be changed. However, the first instruction of the OUTPUT_CHAR subroutine is INT B4 which will allow us to intercept the output string and speed it to the screen.

5) When a BASIC PRINT statement executes, it will generate an INT B4. BASPRINT will service the interrupt, intercept the output string, put it on the screen, then nudge the stack a little so control returns to the OUTPUT_CHAR caller with conditions set to terminate the type A loop.

6) The increase in PRINT speed is a function of the string length to be displayed. A seven-fold increase in speed is demonstrated by the TIMEDEMO program on the CLUBware diskette.

STORAGE REQUIREMENTS:

You can use all of the speedup modules at the same time. Execute each one after booting the system, then run the BASIC Interpreter, execute a BASIC program compiled with or without the /O option, go back to the Interpreter, whatever. Each module uses about 1056 bytes of memory.

Some Limitations:

1) Modules that write directly to the screen buffer will not be sped up.

2) DOS 2.0 pipelining will not pick up the screen output while these programs are running.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

3) WINDOW type packages will probably not work properly.

4) BASPRINT will only work with the IBM versions of BASIC and BASICA. It will not work with any version of BASIC that does not use the IBM ROM routines. This should eliminate all other machines.

Your local users group should have a complete copy of CLUBware Software Diskette #1. If they do not, ask them to request a copy for the programmers in your area. Should you not have access to a local IBM PC users group, send $5.00 and a blank diskette with a self-addressed, stamped, return mailer to:

Rayhawk Automation, N W, Inc.
P.O. BOX 1427
Beaverton, Oregon 97075

We also accept problem reports and requests for enhancements from registered owners. Your name will not be sold or divulged to anyone outside of Rayhawk Automation.

Reprinted below is a sample of just a few of the assembly and BASIC source code routines included on the diskette.

```
COMMENT *

                                CLUBware  (tm)

CLREOL clears the screen from the current position to
the end of the current line. Cursor is left at starting
position.

  Copyright 1984 Rayhawk Automation N.W. Inc
P.O. Box 1427
Beaverton, Oregon   97075

          Algorithm:
                  1) determine current position
                  2) determine number of blanks required
                  3) write that many blanks using INT 10, AH=9


          CALL  CLREOL ( FLAG% )

                  FLAG%       environment flag
                              = 0 means under basic
interpreter
                              = 1 means under compiled basic
                              = 2 means under compiled business basic

  *
;_____________________________________________________________________

;  Normal assembly directives

CODE SEGMENT PARA PUBLIC 'CODE'

          ASSUME  CS:CODE

          PUBLIC  CLREOL

ENV_FLAG          EQU  WORD PTR [BP+6] ; address of environment flag on stack
;_____________________________________________________________________
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
CLREOL    PROC    FAR

          PUSH    AX                      ; save all registers used,
          PUSH    BX                      ;  no data segment local to this
          PUSH    CX                      ;  routine so segment registers
          PUSH    DX                      ;  are untouched.
          PUSH    ES

;         ...     0) load environment flag showing compiled or interpreted

          MOV     BX,ENV_FLAG             ; load address of environment flag


;         ...     1) load current position from 0000:0450 if compiled
;                       or from DS:[0056h] if interpreted

          SUB     AX,AX                ; address system memory
          MOV     ES,AX

          CMP     WORD PTR DS:[BX],0          ; check environment flag
          JNE     COMPILED

          MOV     DX,WORD PTR DS:[0056h]   ; load from basic space
          XCHG    DL,DH                    ; basic has it reversed
          DEC     DL                       ; basic starts count from 1
          DEC     DH                       ;  instead of zero
          JMP     SHORT HAVE_POSITION

COMPILED:
          MOV     DX,WORD PTR ES:[450h]    ; load current position
                                           ;  from system space
HAVE_POSITION:

;         ...     2) determine number of blanks required

          MOV     CX,80                ; first column on line is 0
          SUB     CL,DL                ; subtract off current column
          JZ      AT_END_OF_LINE       ; if zero, then already at end


;         ...     3) write that many blanks using INT 10, AH=9

          MOV     AX,0A20h               ; AH = 10  and  AL = blank
          MOV     BX,0000h               ; BH = 0 (assume always on page 0
                                         ; use existing attribute
          INT     10h                        ; write that many blanks

AT_END_OF_LINE:
          POP     ES
          POP     DX
          POP     CX
          POP     BX
          POP     AX
          RET     2


CLREOL    ENDP

;-------------------------------------------------------------------------

CODE      ENDS

          END
```

```
;-------------------------------------------------------------------------

COMMENT *

                              CLUBware  (tm)

  QPRINT prints a character string on the screen starting
  at the current cursor position.  After the string is
  written to the screen the cursor position is updated
  to just after the string.

  Copyright 1984 Rayhawk Automation N.W. Inc.
                 P.O. Box 1427
                 Beaverton, Oregon   97075

        Algorithm:
                 1) load type of crt display from 0000:463
                 2) load current position from 0000:0450
                 3) load count of characters to write
                 4) load string address
                 5) address the screen segment
                 6) move string to screen segment while synchronizing
                     with horizontal retrace
                 7) update cursor position

        CALL  QPRINT ( FLAG% , CHARACTER$ )

                 FLAG%       environment flag
                             = 0 means under basic interpreter
                             = 1 means under compiled basic not /0
                             = 2 means under compiled basic with /0
                             = 3 means under compiled business basic

                 CHARACTER$  character string to write to screen

                             *
;-------------------------------------------------------------------------

;  Normal assembly directives

CODE SEGMENT PARA PUBLIC 'CODE'

          ASSUME  CS:CODE

          EXTRN   $PUTCPOS:FAR

          PUBLIC  QPRINT

STRING_DESC       EQU  WORD PTR [BP+6] ; address of string descriptor on stack

ENV_FLAG          EQU  WORD PTR [BP+8] ; address of environment flag on stack

;-------------------------------------------------------------------------
```

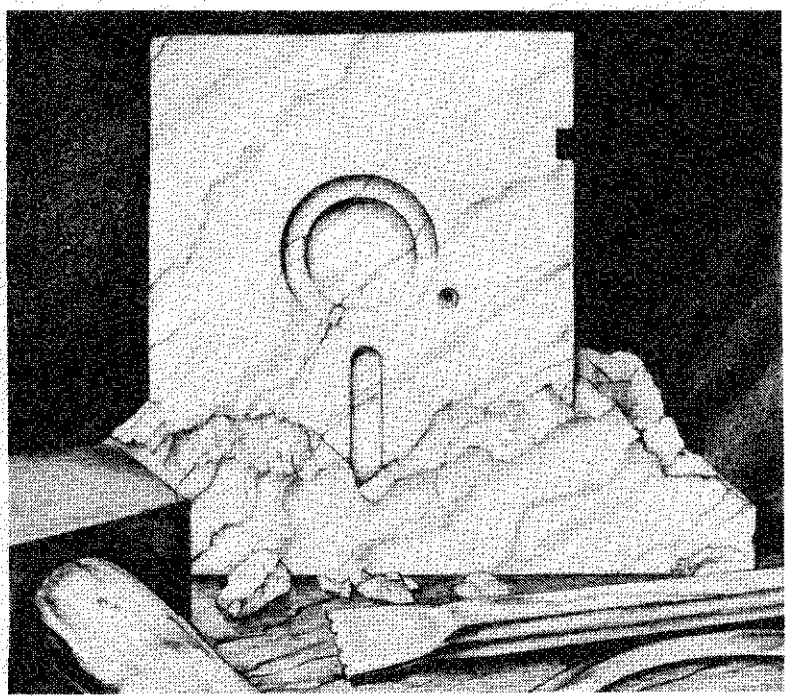

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
QPRINT  PROC    FAR

        PUSH    BP                      ; address parameters on stack
        MOV     BP,SP

        PUSH    AX                      ; save all registers used,
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    DI
        PUSH    SI
        PUSH    ES

;       ...     0) load environment flag showing compiled or interpreted

        MOV     SI,ENV_FLAG         ; load address of environment flag
        MOV     SI,WORD PTR [SI]    ; load flag itself


;       ...         1) load type of crt display from 0000:463

        SUB     AX,AX                   ; address system area
        MOV     ES,AX

        MOV     DX,WORD PTR ES:[463h]   ; load address of display adapter
        ADD     DX,6                        ; address crt status port


;       ...     2) load current position from 0000:0450 if compiled
;                    or from DS:[0056h] if interpreted

        CMP     SI,0                            ; check environment flag
        JNE     COMPILED

        MOV     CX,WORD PTR DS:[0056h]  ; load from basic space
        XCHG    CL,CH                   ; basic has it reversed
        DEC     CL                      ; basic starts count from 1
        DEC     CH                      ;  instead of zero
        JMP     SHORT CALC_POSITION

COMPILED:
        MOV     CX,WORD PTR ES:[450h]   ; load current position
                                        ;  from system space
CALC POSITION:
        SUB     AH,AH           ; isolate row number in AX
        MOV     AL,CH
        MOV     BL,80           ; multiply row by 80 bytes per row
        MUL     BL
        SUB     CH,CH           ; add in column number
        ADD     AX,CX
        MOV     DI,AX
        SHL     DI,1                ; multiply by 2 to account
                                    ;  for attribute bytes

;       ...     3) load count of characters to write
;                  only 1 byte if interpreted
;                  2 bytes if compiled

        MOV     BX,STRING_DESC      ; load address of string descriptor

        CMP     SI,0                ; running compiled?
        JNE     NOT_INTERPRETED

        MOV     CL,BYTE PTR [BX]    ; load 1 byte count itself
        SUB     CH,CH
        INC     BX                  ; bump to string address
        JMP     SHORT LOAD_ADDRESS

NOT_INTERPRETED:
        MOV     CX,WORD PTR [BX]    ; load 2 byte count itself
        ADD     BX,2                ; bump to string address

LOAD_ADDRESS:

;       ...     4) load string address
;                    next two bytes if normal compiled
;                    three bytes past if in business basic

        MOV     BX,WORD PTR [BX]    ; load offset of string

        CMP     SI,3
        JNE     NOT_BUSINESS

        ADD     BX,3                ;  Support BBC String format

NOT_BUSINESS:

        MOV     SI,BX               ; move offset over
        MOV     BH,07h              ; use normal video attribute


;       ...     5) address the screen segment

        MOV     AX,0B000h           ; screen seg for monochrome card
        CMP     DX,03DAh            ; is this a graphic card?
        JNE     MONOCHROME

        MOV     AX,0B800h           ; load screen seg for graphic card
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
MONOCHROME:
          MOV     ES,AX               ; address the screen buffer

;         ...     6) move string to screen while synchronizing
;                     with horizontal retrace

DISPLAY_LOOP:
          LODSB                       ; load next character
          MOV     BL,AL

          CLI
HSYNC_WAIT1:
          IN      AL,DX               ; check for horizontal retrace
          TEST    AL,1
          JNZ     HSYNC_WAIT1         ; wait for retrace
HSYNC_WAIT2:
          IN      AL,DX               ; check for horizontal retrace
          TEST    AL,1
          JZ      HSYNC_WAIT2         ; wait for retrace

          MOV     AX,BX
          STOSW                       ; store character and attribute
          STI

          CMP     DI,4000
          JL      NOT_OFF_PAGE
          MOV     DI,0                ; start at top again
NOT_OFF_PAGE:

          LOOP    DISPLAY_LOOP        ; repeat cx times


;         ...     7) update cursor position

          MOV     AX,DI               ; load current position
          SHR     AX,1                ; discount attribute bytes
          SUB     DX,DX               ; no sign in this division
          MOV     BX,80               ; divide by nmbr of chars per row
          DIV     BX
          MOV     DH,AL               ; store row number
                                      ; column number already in DL
          MOV     BH,0                ; assume page 0
          MOV     AH,2                ; request new position
          INT     10h

          XCHG    DH,DL               ; basic likes it reversed
          INC     DL                  ; basic starts count from 1
          INC     DH                  ;  instead of zero
          MOV     SI,ENV_FLAG         ; load address of environment flag
          CMP     WORD PTR DS:[SI],0  ; examine flag
          JE      NOT_COMPILED

          CMP     WORD PTR DS:[SI],3  ; business basic?
          JE      RET_TO_BASIC

          CMP     WORD PTR DS:[SI],2  ; compiled with /O?
          JE      WITH_SLASH_O
                                                                  ; FLAG = 1
          MOV     WORD PTR DS:[87h],DX ; store screen address
          JMP     SHORT RET_TO_BASIC

WITH_SLASH_O:                                                     ; FLAG = 2
          MOV     DI,SEG $PUTCPOS     ; address put cursor routine
          MOV     ES,DI
          MOV     DI,OFFSET $PUTCPOS
          MOV     DI,WORD PTR ES:[DI+2] ; load offset of cursor storage
          MOV     WORD PTR DS:[DI],DX ; store new cursor position
          JMP     SHORT RET_TO_BASIC

NOT_COMPILED:                                                     ; FLAG = 0
          MOV     WORD PTR DS:[0056h],DX ; store new cursor position

RET_TO_BASIC:                                                     ; FLAG = 3

          POP     ES
          POP     SI
          POP     DI
          POP     DX
          POP     CX
          POP     BX
          POP     AX
          POP     BP
          RET     4

QPRINT    ENDP

;_____________________________________________________________________

CODE      ENDS

          END                                                     □
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



"In the art of programming the difference between greatness and mediocrity is often the quality of the artist's tools."

In every top programmer's tool kit there are usually a few secret weapons. PolyMake and PolyLibrarian have become the newest essential secret weapons for hundreds of professionals who are setting new standards of productivity and performance for themselves and the industry.

Praise From Professionals

"PolyLibrarian is a powerful tool for serious hobbyists and professional programmers. It is a thing of beauty; a work of art. The documentation is excellent, the large selection of commands are intuitive to learn and easy to use, and the program itself provides a service of inestimable value. It is refreshing to see a tool of this caliber available for serious programmers. The design of this program shows foresight and ingenuity. It sets the human engineering standard for programming utilities. If all of a programmer's tools were so simple to work with, better programs could be developed in less time and with fewer headaches. *Bravo, Polytron!* I look forward to your next product."

Dan Rollins,
Programmer and Author writing in PC Age

"PolyLibrarian is an extensive, friendly utility . . . an excellent tool for serious programmers. It combines professional quality with a flexible user interface."

Greg Estes
Editor, Programmers Journal

"I am thoroughly pleased.[PolyLibrarian] is definitely one of the best products of its type that I have ever used on any system."
An Unsolicited Comment from
Steve Kauffman
Consulting Engineer

To Order or Request Literature Call

1-800-547-4000

Ask For Dept. 31
VISA & MasterCard Accepted
Or Send Checks, P.O.'s To:
Polytron Corporation DS-310-F1
P.O. Box 787, Hillsboro, OR 97123
Add $1.25 Shipping Charge For Each Product Ordered.

POLYLIBRARIAN

The Object Module Library Manager for MS-DOS™

PolyLibrarian lets you create a single library from related Objects Modules. The IBM PC Linker will then select only the modules necessary to produce an executable file. PolyLibrarian can: ☐ Create, organize, reconstruct, dissect, and compress libraries of object code modules. ☐ Add, delete and replace an object module. ☐ Extract a module from a library to create an OBJ file. ☐ Change public and external names. ☐ Optimize a module to shorten overall size of a library. ☐ Modules are time-stamped within the library. ☐ Supports full path names. ☐ Supports Microsoft's original and latest Linker (Version 2.4). ☐ Includes installation program to customize options to your specific needs. ☐ Features three Modes of Operation (Interactive, Command File, Command Line) that can be freely intermixed. ☐ Surpasses performance of other primitive librarians. ☐ Features a highly functional user interface that uses simple meaningful key words. ☐ Requires 128K RAM, MS-DOS (PC-DOS) 1.1, or 2.0. ☐ Compatible with any compiler or assembler that uses the MS-DOS (PC-DOS) Linker. ☐ Shipped with comprehensive Users Manual including step-by-step tutorial for **$99**

POLYLIBRARIAN II

Extends PolyLibrarian's power to Intel™ Format Libraries.

PolyLibrarian II is an enhanced version of PolyLibrarian. In addition to all the features and functions listed above for PolyLibrarian, this version adds a new dimension of productivity for programmers using Intel Language Products. ☐ Compatible with any Intel Format Library and any Microsoft Format Library. ☐ Compatible with Intel's Link 86 under MS-DOS. **$149**

POLYMAKE

The Intelligent Program Builder & Maintenance Tool for MS-DOS.

PolyMake frees you from the need to remember which files depend on others and which files have been modified. PolyMake will: ☐ Remember the exact sequence of operations necessary to make a new version of a program. This will significantly reduce development time, prevent bugs & manage large software projects ☐ Automatically invoke your compiler, assembler, linker, librarian, or do whatever is necessary to bring all dependent files up to date. ☐ Compare the date and time of all relevant files and use internal rules to rebuild a program or complete software system. ☐ Rules can be modified and expanded. ☐ Extensive debug capabilities help construct efficient make files. ☐ Includes features superior to UNIX Make. ☐ Fully automatic operation under MS-DOS 2.X. ☐ Full path name capabilities under MS-DOS 2.X. ☐ Requires 128K. Shipped with a comprehensive Users Manual **$99**

TM

POLYTRON

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HEADER RECORDS IN RANDOM ACCESS FILES (Part II)

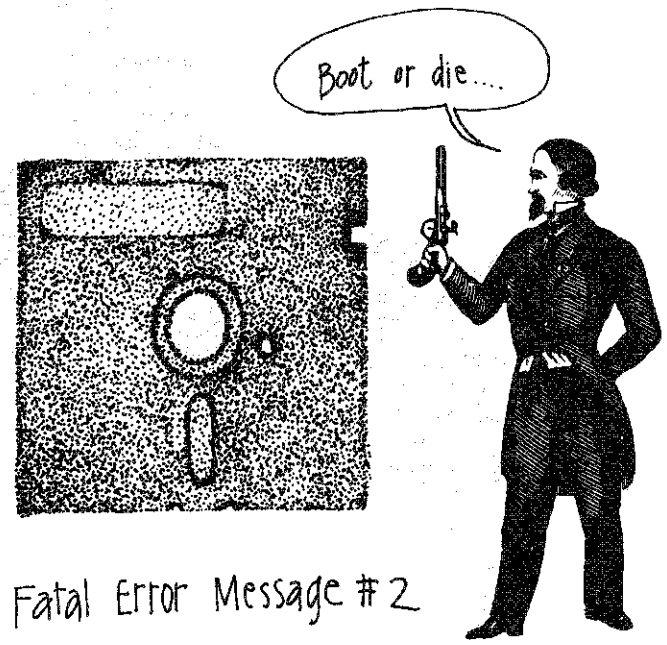
by Ronald Moak

Part I of this article discussed most of the BASIC commands used to create and use random access files. Part II will use those commands and concepts to create a random access file with special attributes. The file we will generate this time will contain two different record types. One record type will be the data record.

Often when writing application programs that utilize random access files we need to store additional information concerning the contents of the file. This additional information could be the number of records in the file or a series of fields describing the data types, last record access, range of values, etc.

One place to store all of this additional information is in a separate file. Usually, this file has the same name as the main data file, but with a different extension. This method is fine if there is a great deal of information to be stored about each file. Conceptually, it is easier to write routines for two separate files. In general, a separate summary file is often opened and read by a program, then the main data file is opened for processing.

The primary problem with two files is the burden placed on both the program and the end user. Every time the data file is read, the program must check to be sure both files are present. If one of the files is missing, the user must be notified of the missing file. This forces the user to be certain that both files are copied when transferring data from disk to disk. If for some reason the summary file gets deleted accidentally then the main file is next to useless, unless you have made provisions to allow recreation of the data.



Special header records can take the place of summary files by storing information in the main data file. When you combine these two files you no longer have to track multiple files.

Header records may be simple or complex depending upon how much information you need to remember. The following listing illustrates a very simple header record. The length of the header record is much less than the length of the normal data record, so 124 bytes of "filler" is needed to pad out the rest of the first record (128 bytes must be read each time the file is accessed).

Listing 1.

```
1000    OPEN FILENAME$ AS #1 LEN=128
1010    FIELD #1, 4 AS RNUMREC$, 124 AS RFILLER
1020    FIELD #1, 20 AS RNAME$, 20 AS RCOMPANY$,
        20 AS RSTREET$, 20
        AS RCITY$, 2 AS RSTATE$, 9 AS RZIP$, 20 AS
        RCONTACT$, 17 AS
        FILLER
```

The FIELD statement in line 1010 defines the map, or data mask, for the header record. In this case, the special header record only stores the total number of records in the file. The only tricky thing about this special header type is the need to always remember that the data starts with the second record and not the first.

Things tend to get pretty sticky when the information stored in the header occupies more bytes than the length of the standard data record. Also, for example, what happens if the information in the header is actually a fixed number of similar records that describes the values in the data record? With a little creative programming you can store any amount of information in the header.

The following listing is a subroutine from a program that that plots data stored in a random access file. The file is composed of two parts, the header and the data. In this example we open the file, read in the contents of the header, close the file, and finally open the file again to be accessed for the data. We could have opened the file twice with different file numbers and different lengths, but in this case we chose to first open and then close the file.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Listing 2.

```
10      DIM MAX! (10),MIN!(10),TITLE$(10)
995     ******************************************************
996     *
997     *     Read header file and data
998     *
999     ******************************************************
1000    OPEN FILENAME$ AS #1 LEN=28
1010    FIELD #1, 4 AS RMAX$, 4 AS RMIN$, 20 AS RTITLE$
1020    FOR I% = 1 TO 10
1030        GET #1,I%
1040        MAX!(I%) = CVI(RMAX$)
1050        MIN!(I%) = CVI(RMIN$)
1060        TITLE$(I%) — RTITLE$
170     NEXT I%
1080    CLOSE #1
1095    ******************************************************
1096    *
1097    *     Open data file
1098    *
1099    ******************************************************
1100    OPEN FILENAME$ AS #1 LEN=40
1110    FIELD #1, 4 AS VAL1$, 4 AS VAL2$, 4 AS VAL3$,
        4 AS VAL4$, 4 AS VAL5$, 4 AS VAL6$, 4 AS VAL7$,
        4 AS VAL8$, 4 AS VAL9$, 4 AS VAL10$
1115    ******************************************************
1116    *
1117    *     Calculate offset
1118    *
1119    ******************************************************
1120    OFFSET% = ((10 * 28) / 40) + 1
1130    RETURN
```

The header record in listing 2 required 280 bytes or 240 more bytes than the length of the data record. Also, the header record is composed of 10 separate records that contain information about the data. In this example, the data portion of the file contains values to be plotted. Information in the header part of the file contains the maximum and minimum values for each of fields in the file. The header also holds a 20 character title for each field.

An offset value is calculated to determine the position in the file of the first data record. The offset value is composed of the length of the header (10 records * 28 bytes/record) divided by the length of the data record (40 bytes) plus 1 for insurance. In the above case we could have left off the 1 because the result of the header divided by the data was a whole integer. However, in most cases the 1 provides insurance against integer rounding.

Any future reference to data records within the file will have to be made by adding the record number to the offset. This will insure that no attempt is made to read the data in the header as data to be plotted.

As you can see by the previous two examples, using a header record to store additional information that describes the file is not at all that difficult. The inconvenience of keeping track of an offset to the first data record is negligible compared to the overhead of multiple files or forcing the user to do extra work. □

TECHNICAL DISK MAGAZINE

- As editor (alias Bill Salkin) of **PC FIRING LINE/PC UNDERGROUND,** the bi-monthly disk magazine for the IBM PC, I invite you to join us as we discuss: **critical error-handling routines** random file I/O, DOS 1.1 and 2.x sequential file I/O, different memory models, **maze-generations programs**, bicubic splines, drool over "printf" source code, share **FREEWARE**, provide programmer-oriented **DEMOS** and tips, dissect utilities like the DOS 2.x **EXEC function**, continue our **ADA, ASM, C,** and **FORTH tutorials, and more! Sizzling ready-to-use source code** included in each jam-packed issue (currently 2 DS/DD disks per issue.)
- *Feel the thrill and power of our pure assembly language main driver — no speed limit on this magazine. All articles* compiled without errors! And under DOS 2.x you can even execute DEMO programs from within PCFL/PCUG.
- Price? A moderate $12 per issue, or $72 for a one-year (six-issue) subscription. Make checks payable to ABComputing. (All currency in U.S. dollars. Foreign countries send money orders and add $5 air mail for each issue.)
- Make non-profit copies for your friends. Those receiving these copies, are asked to pay $6 to help defray production costs. Make checks payable to ABComputing.
- Requires 128K RAM, any release of DOS, one double-sided disk drive, and a good sense of humor. Note: We ship ONLY DS/DD disks.

ABComputing
P.O. Box 5503, North Hollywood, CA 91616-5503
(818) 509-9002

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

RTOOL, A PC-RESIDENT PROGRAMMER'S TOOLKITp10

Review by Jim Gore

The power of our personal computers continues to grow. Productivity tools are constantly surfacing that put new capabilities at our fingertips. The IBM Personal Computer has inspired some very creative programming efforts and there is no end to the flow in sight.

Witness the growing list of software "utility" programs that stay resident as long as your computer is turned on. When you install these utilities with another application, you have access to both programs concurrently. The resident software adds a dual-tasking mode to the PC. And if the resident tasks are useful, they can make your computer time more productive.

One of these resident programs is RTool from BitWit Software. RTool is a set of general purpose functions that is installed once and can be invoked at any time, even while running another program. The operations provided are organized into four modes: a full screen scratchpad editing mode, DOS mode, Macro mode, and a Help mode that facilitates extensive on-line reference file capabilities.

HOW IT WORKS

RTool is usually installed when you first boot up your computer, after other DOS resident software such as RAM disks, spoolers, and mode commands are loaded. The Autoexec.bat file in Figure 1 sets up a RAMDISK, sets the system time and date, enables graphics print screens, and then loads RTool with 10 blank scratchpads and two named on-line reference files.

Figure 1. A sample Autoexec.bat file

```
. autoexec.bat file

QDXT 8/A
PWRUPCLK
GRAPHICS
RT2 -S10 -X -HB:RTOOL.HLP -HB:ADVDOS.HLP
```

The last line of this BAT file is the RTool invocation.

RTool for DOS 2.0 takes up about 35,000 bytes plus 1640 bytes per scratchpad (you can have up to 20 scratchpad pages) of system memory.

When RTool is loaded it sets up all the necessary control logic and transparently assumes control of the PC keyboard and the DOS/Program interface. As you desire, you can awaken RTool with a special keystroke (Ctrl-Alt plus S,D,M, H or Enter). Rtool saves the PC's "state" and sets up its own interface, giving you access to the resident tools. If a program is executing when you invoke RTool, it is "bookmarked" and put on hold.

While in an RTool mode, commands are invoked by pressing a Function key. The bottom five lines on the screen display a status menu showing the active mode and the currently active function keys. While in RTool, keystrokes are conveniently minimized by a clear use of special keys. When you exit Rtool, the state of your computer is restored, and the screen appears exactly as it did when you left. A bookmarked program continues execution.

The program is generally quite useful, written as a tool to fit many situations. It takes very little time to learn its basic functions, and the more you use it the more uses you discover for it. If nothing else, the 1600 character type-ahead buffer which RTool sets up lets you maintain a brisk work pace on your computer. But that's just the beginning of this tool's power.

SCRATCHPAD MODE

The Scratchpad mode provides several useful operations, including the transfer and storage of (text) screen images, full screen text editing, and text file retrieval. A scratchpad page holds 20 80-character lines of text. In scratchpad mode, only standard ASCII characters are recognized; the PC's text graphic characters are ignored. This mode reduces (and in some cases eliminates) the need for hard copy program listings, and handwritten notes during program development. You can run programs under DOS or DEBUG and have access to source code files and a text editor concurrently.

I find the scratchpad mode quite useful during the debugging of compiled Basic programs. While running a complex program, I often invoke scratchpad mode to SCAN a source code file, sometimes saving it in scratchpads for later reference.

This way I can refer to code when I need to, and make modification notes to myself with RTool's full screen page editor, just by putting the executing program on hold. If I modify any code, I can save the modified pages to an ASCII disk file and later read them with my usual text editor.

This saves me time, and takes paper and pencil operations out of the program debugging loop. Program documentation

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

has also become easier to manage. I can replace the endless flow of handwritten documentation notes with scratchpad files, and organize them easily by using designated file names/extensions. Spontaneous bursts of inspiration can be recorded and saved on the spot. Scratchpad mode provides a very convenient way of documenting programs and program features. The XFER IMAGE option captures the first 20 lines of the previous screen into a scratchpad, which can then be edited or saved as an ASCII disk file. While working on an application containing several levels of menus, I used the XFER IMAGE command to construct a file containing all the menus in the program. Creating the same file by hand would have meant considerable typing; RTool allowed me to do it in a couple of minutes.

DOS MODE

Among the flashier features offered in RTool's DOS mode is the ability to terminate a currently running program, even if control break is not otherwise enabled. This could be handy if you tend to get into a lot of infinite loops or other "hang-up" situations.

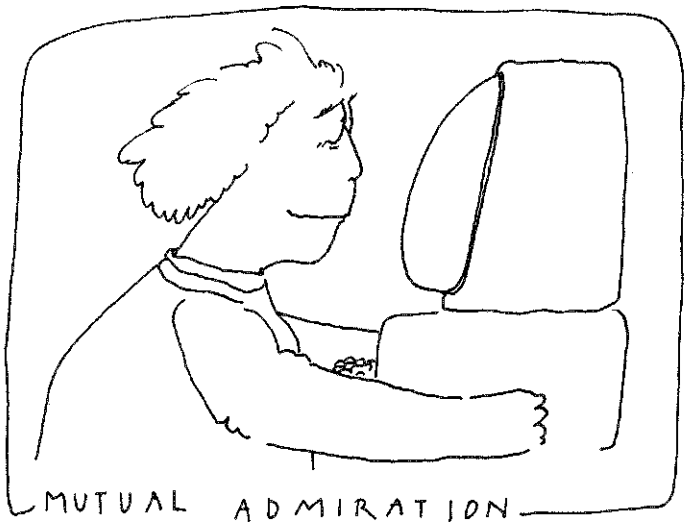
DOS mode also lets you rename and delete files, keeps track of disk space used and space available, lets you change directories (DOS 2.0 only), do full or partial directory listings with variations of the dir command, and will transfer directory listings into resident scratchpads for later reference. All in the middle of running another program.

MACRO MODE

Macro mode adds extensive macro capability to your computer without conflicting with programs that have their own macro facilities. Macro mode provides a batch job operation that lets you transfer whatever is displayed in a scratchpad page into the keyboard buffer. You can use this feature to conveniently transfer data between programs, or transfer memory resident text directly into a text editor.

Macro mode has many more uses. Macros can be stored in a file and re-loaded when needed. Macros can be assigned to a keystroke (ALT-SHIFT- plus A-Z) and submitted directly without invoking RTool. Whatever text is stored for that macro will be appended to the keyboard buffer. RTool lets you set up a "trap" macro which, while turned on, captures every keystroke made while running another program. This enables you to "capture" and later save a macro while observing each keystroke's effect on the running program. Then you can repeat the entire process anytime just by invoking the macro that you just trapped.

Macros can also be submitted inside RTool. When you enter Macro mode, the default operation is macro select. You cursor through the macro list to choose an existing one, or



press the ADD MACRO key to add one. Before a macro is submitted, you can edit it if you wish.

Macros can be assigned tags and can be nested to perform complex batch operations.

HELP MODE

This is perhaps where RTool shows its greatest potential. If you like on-line reference facilities, this program is worth taking a look at. RTool comes with three ready-to-use help files (RTool help, Advanced DOS topics, and MASM, DEBUG, and LINK help). It also includes programs to generate your own help files, and create permanent on-line reference systems.

RTool help files are organized on a "topic" basis. A single file can include up to 54 topics, with a topic having 1 to 6 pages of associated text. Help screen pages can be easily transferred to scratchpad space for even faster retrieval. While in Help mode, the current topics for a loaded help file are displayed in a menu-like list (Figure 2). Topic selection is made by cursoring through the list of topics, and pressing the Enter key. Once in a help topic, you can use the PgUp/PgDn keys to page through information about the chosen topic.

HELPGEN is a file editor that can be run over Rtool to edit existing help files and create new ones. HELPGEN prompts you for a topic name and then lets you construct or edit help pages for that topic. While in HELPGEN, you still have access to all the resident operations, and can use them to SCAN other files and transfer text into help pages. Screen images stored in scratchpad space can be batch submitted to the keyboard and routed into the waiting help file page.

With the HELPGEN program, virtually any type of manual that you frequently refer to can be stored on-line. Language syntax, custom applications, documentation and other reference files

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

YOUR CODE MAY BE WASTING ITS TIME!
THE PROFILER™ CAN HELP . . .

- **Statistical Execution Profiler**
- **Works with any language**
- **Completely configurable**
- **Up to 16 partitions in RAM/ROM**
- **Time critical code optimization**
- **Abnormal code behavior tracking**
- **Graphic presentation of results**
- **Easy to use menu interface**

THE PROFILER is a software package which gives you, the programmer, a powerful tool for locating time consuming functions in your code and allows you to performance tune your program. With **THE PROFILER** you can determine where to optimize your code for maximum benefit, then measure the results of your efforts.

Using **THE PROFILER**, you can answer questions like:
Where is my program spending its time?
Why is my program so slow? What is it doing?
Is my progam I/O bound? CPU bound? Are data buffers large enough?
How much improvement did my changes make?

THE PROFILER is completely software based and consists of a system resident driver and a monitor program. The memory partitions can range from 1 byte to 1 megabyte in size and can be anywhere in the address space.

NO ADDITIONAL HARDWARE IS REQUIRED!

Requires an IBM PC or compatible system with a minimum 64k and one drive.

THE PROFILER is available for $125.00 **(New Low Price)** from DWB Associates or ask your software dealer. To order or for more information, call or write DWB Associates. VISA/MC accepted. Dealers welcome.

IBM is a trademark of IBM Corp. MSDOS is a trademark of Microsoft Corp.
THE PROFILER is a trademark of DWB Associates.

dwb
Associates
P.O. Box 5777
Beaverton, OR 97006
(503)629-9645

Introducing the Creative Genius. . .

YOU. **You have the system . . . and now you have the means. DataBurst™, a genuine full-screen processor and BASIC program generator, brings a System/34 interactive environment to your IBM® -PC. DataBurst™ provides screen design and program generator capabilities similar to SDA on the System/34. The unique DataBurst screen editor provides fast, easy screen design. Program independent screen formats reduce both development and maintenance time.**

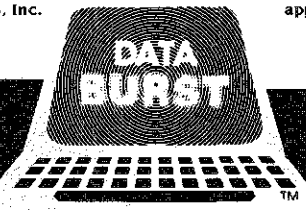
During execution of your program, DataBurst™ handles all of the screen and keyboard operations through one Assembly language module, requiring as little as 12K of memory. A true full-screen processor, DataBurst™ provides you with the power of System/34 SDA on your IBM® -PC.

DataBurst™ is available through your local computer retailer or direct from Key Solutions. To order directly, please send check, money order or purchase order for $225* to Key Solutions, Inc., P.O. Box 2297, Santa Clara, CA 95055. BASIC Compiler Runtime Support is available for an additional $40.* (Please inquire about the release dates for additional language interfaces.)

IBM® -PC is a registered trademark of IBM Corporation.
DataBurst™ is a trademark of Key Solutions, Inc.

*In California add applicable sales tax.

The Design Tool for the Creative Programmer



KEY SOLUTIONS

can be easily constructed and are then available at a keystroke when you really need them. RTool help files are Huffman encoded for maximum space efficiency, and can only be edited with the HELPGEN program. This presents little inconvenience however, since the HELPGEN editor uses the same commands as the scratchpad editor and is quite simple to use.

LIMITATIONS

RTool is not without its limitations. It is not compatible with keyboard enhancement programs such as PRO-KEY, and not compatiable with mouse driven systems that reset the keyboard logic. RTool may not work on all DOS 2.0 networks, and will not work with multi-tasking/multi-job DOS enhancement programs. RTool requires PC-DOS 1.1, 2.0, or 2.1, and will work on selected PC-compatibles running PC-DOS (but not MS-DOS).

RTool comes with a program RTCHECK to diagnose your system configuration and warns about incompatibility problems. This program is available on a demonstration disk so you can test it out on your system before purchasing it.

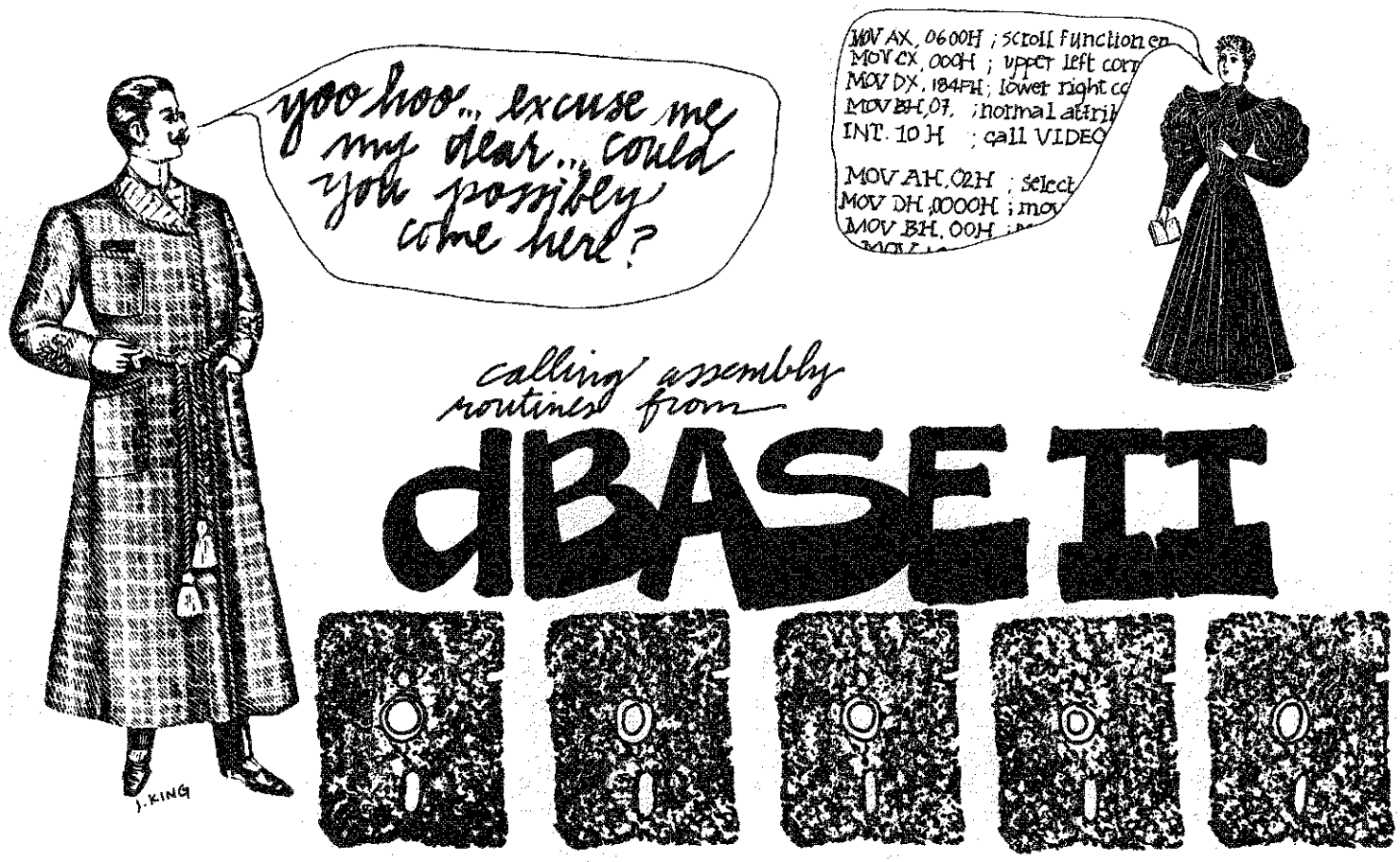
RTool ($159)
BitWit Software
3790 El Camino Real, No. 205
Palo Alto, CA 94306
(415) 858-0120

Figure 2. RTool help file topics

Intro to RTool	Macro Tags
Intro to HelpGen1	Macro Trap
Intro to FileGen	Macro Editing
Installing RTool	Macro Addition
Accessing RTool	Batch Job
Modes	Clear Kbd
Screen Design	Clear Mode
Text Editing	Macro File I/O
RTool Files	Help Files A/B/C
ScanFile/AutoLoad	Load Help File
Transfer Image	Viewing Help Topics
S/Pad Disk I/O	Transfer Screen
Directory	RTool Errors
Rename File	Main Mode, HelpGen
Delete File	Edit File, HelpGen
Change Dir	Edit Topic, HelpGen
Kill Program	Using FileGen
Macro Submission	Helpfile Text hints
RTool Quick Ref	

□

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED


yoo hoo.. excuse me my dear.. could you possibly come here?
MOV AX, 0600H ; scroll function en
MOV CX, 000H ; upper left corn
MOV DX, 184FH ; lower right co
MOV BH, 07 ; normal attrib
INT 10 H ; call VIDEO
MOV AH, 02H ; select
MOV DH, 0000H ; mov
MOV BH, 00H ;
J. KING

calling assembly routines from

dBASE II

a review by warren hillman

CALLING ASSEMBLY ROUTINES FROM dBASE II

by Warren Hillman

dBKey ($49)
Metronomics
P.O. Box 24230
New Orleans, LA 70184
(504) 488-2266

Bill McNary of New Orleans has created a very functional tool for dBase II programmers called dBkey. He has written a number of highly desirable routines for dBase II. The routines are assembly language modules and are poked into memory. The documentation and programs clearly show you how to develop your own assembly routines and put them into the appropriate memory locations. dBase II's CALL function gives you the hook to access them once they are loaded.

As dBkey's documentation states, it is "a tool for creative programming." You are no longer limited to the arbitrary restrictions Ashton-Tate has set up. Whatever you can modularize into a short assembly routine can become a part of a dBase II program using dBkey's straightforward approach.

dBkey offers a wide variety of functions. You can write to and scroll windows, blank a user defined window, determine the default disk drive, find the total free disk space, place DOS commands into the keyboard buffer and exit to them, and make sounds and write music, as well as read the system date and time. Bill mentioned that one programmer included the company song in his application, while another wrote Christmas carols. Bill is working on dBase III compatibility. There should be no problem once their dBase III's CALL function is implemented (which currently is non-functional).

Reprinted here is an example of dBkey's 18 programs. This one determines the total space available on diskette and tells you how much is left. A necessity in many applications.

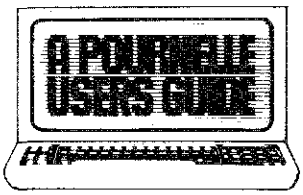
[Editor's note: Metronics is in the process of doing a name search and will be changing its name to something other than dBkey. You may still request the product under its old name at the same address.]

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
*************************  DISK.PRG  ************************************
*                       (C)Copyright 1983 Bill McNary
*                          All rights reserved
**************************************************************************
* This program will return the free space and the total space from the drive
* selected by the user. The user must initialize the variable MDRIVE before
* calling the routine. MDRIVE is the drive to be checked. The procedures
* return  the results in the following memory variables:
*         - MFREE:  contains the number bytes in free space on the disk
*         - MTOTAL: contains the number bytes of total disk space
**************************************************************************

* Use case statement for branching under various conditions of the variable
do case
case type(MDRIVE)='U'
        ? 'MDRIVE must be initialized prior to the call to DISK'
        ? 'Please enter the drive letter and try again.'

case type(MDRIVE)='L'
        ? 'MDRIVE must be a character. The variable must be reset.'

case type(MDRIVE)='N'
        ? 'MDRIVE must be a character. The variable must be reset.'

case type(MDRIVE)='C'


        * Convert drive to ASCII input string
        do case
        case !(MDRIVE)='A'
                store 65 to MDRIVE:IN
        case !(MDRIVE)='B'
                store 66 to MDRIVE:IN
        case !(MDRIVE)='C'
                store 67 to MDRIVE:IN
        case !(MDRIVE)='D'
                store 68 to MDRIVE:IN
        case !(MDRIVE)='E'
                store 69 to MDRIVE:IN
        case !(MDRIVE)='F'
                store 70 to MDRIVE:IN
        endcase

        * Enter the machine code
poke 61678,30,128,38,122,241,15,144,144,144,144,144,144,144,144,144
poke 61693,144,144,144,144,144,144,144,144,144,144,144,144,144,144
poke 61706,144,144,144,144,144,138,22,122,241,180,48,205,33,60,2,115,81
poke 61723,180,28,205,33,180,0,135,202,81,247,226,80,51,210,51,192,191,3,0
poke 61742,189,3,0,139,49,247,197,1,0,117,9,209,238,209,238,209,238,209,238,71
poke 61762,129,230,255,15,117,1,64,71,69,226,228,139,236,139,94,4,142,219,89,81
poke 61782,247,225,163,127,241,137,22,129,241,90,88,247,226,163,123,241
poke 61798,137,22,125,241,31,195,180,54,205,33,50,228,82,247,225,80,147
poke 61815,235,212,65,0,0,0,0,0,0,0,0,0,0

        * Enter the input parameters
        poke 61818,MDRIVE:IN

        * Structure the call
        store ' ' to CHECK
        set call to 61678
        call CHECK
        release CHECK,MDRIVE:IN

        * check for a disk read error
        store peek (61827) to MERROR
        if MERROR>0
                store 'ERROR' to MTOTAL
                store 'ERROR' to MFREE
                return
        endif
        release MERROR

        * Move the output into memory variables
        store peek (61819)+(peek (61820)*256) to MTOTAL
        store MTOTAL+(peek (61821)*65536)+(peek (61822)*16777216) to MTOTAL
        store peek (61823)+(peek (61824)*256) to MFREE
        store MFREE+(peek (61825)*65536)+(peek (61826)*16777216) to MFREE
endcase
release MTEST

* return to caller
return
**************************************************************************
* The last lines to prevent the loss of executable instructions
**************************************************************************
```

□

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



TWO NEW POURNELLE USERS GUIDES FROM BAEN COMPUTER BOOKS

The newest in a continuing series commissioned, selected, edited or approved by Jerry Pournelle—books you can depend on by a name you can trust!

THE ESSENTIAL USERS GUIDE TO THE IBM PC, XT, AND PCjr

by Dian Girard

Of *course* you want an IBM—
but *which* IBM?

READ THIS BOOK

This complete and easy-to-understand guide to the IBM personal computer family includes an Appendix with extensive technical details plus programs that can be keyed in and are also available on disk!

NOVEMBER 55922-2 • 256 pp. • $6.95

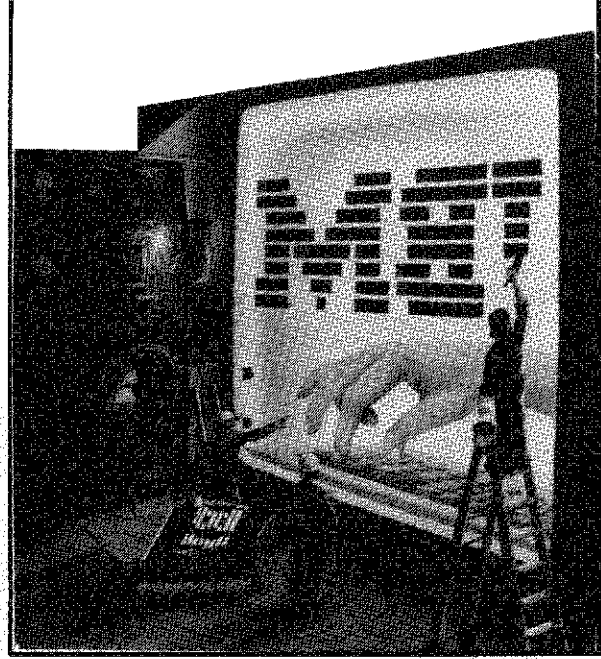
THE USERS GUIDE TO CP/M SYSTEMS

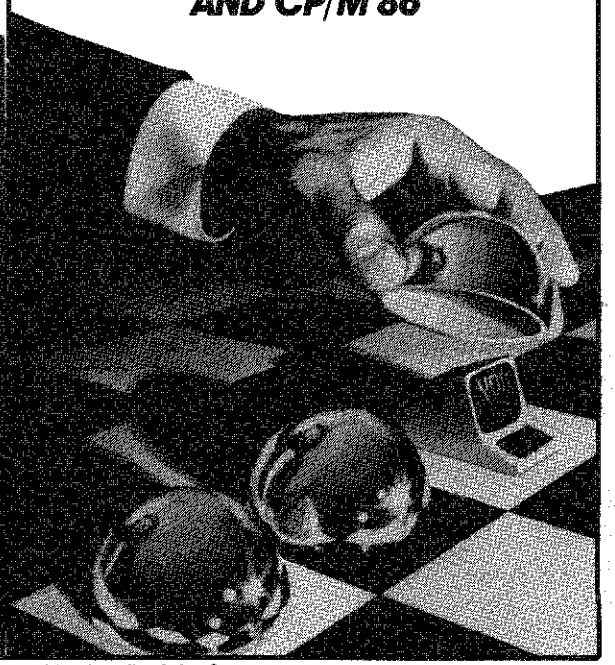
Tony Bove and Cheryl Rhodes

Finally! An *understandable* training manual that offers hands-on instruction for the beginner. Newcomers to the world of personal computing need *friendly* help, not confusing instructions that require decoding before they can be applied. This handbook, written by the editors of *Users Guide to CP/M* magazine, was specifically designed for the novice who wants to dive right in and learn while doing. Provides in-depth instruction in two crucial areas: word processing and programming.

NOVEMBER 55921-4 • 224 pp. • $6.95

INCLUDES CP/M PLUS AND CP/M 86





Distributed by Simon & Schuster Mass Merchandise Sales Company
1230 Avenue of the Americas • New York, N.Y. 10020

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LINK UP TO PASCAL'S POINTER POWER

by Chet Floyd

Every program presents the need to organize data in some way, and programming languages offer various means of doing this. Variables, record structures, and arrays come quickly to mind as simple and efficient techniques. As new hardware and software developments extend the power of microcomputers, and therefore the uses we can program them for, more powerful ways to organize data need to be exploited.

In applications that must dynamically manage data or text, the linked list is an outstanding organization technique. Using linked lists, Pascal can efficiently perform dynamic tasks such as organizing the fields on data entry screens, organizing and relating records or text strings, building full and partial index lists, and many more tasks. Linked lists can be rapidly created, maintained, and discarded with little overhead. Any logical order of the physical data can be provided by the list. Lists can be used to define subsets and supersets of the underlying data. These characteristics of efficiency and flexibility make lists very useful in dynamic and complex applications. The key to list efficiency is that the underlying data, that the list controls, does not have to be moved.

Most Pascal textbooks dedicate a chapter, or most of one, to list processing. The basic terminology and the several types of lists and methods of linking them are discussed. By focusing this article on one kind of list (the pointer-linked doubly-linked list) we can provide what most texts do not: specific code examples that maintain lists. By focusing on Microsoft's implementation of Pascal 3.13, ambiguities caused by Pascal's notorious dialect problems will be avoided. IBM Pascal 2.0 and MS Pascal 3.2 will be considered equivalent to 3.13 for this discussion.If you have another compiler, you should be able to easily make the required conversions.

To illustrate Pascal's list management techniques, imagine an application that uses a master record that has an arbitrary number of variable length records attached to it. Perhaps this data might be a recruiting firm's personnel resume file. The examples will split this into two lists, one doubly-linked circular and one linked to the first. Double linking is chosen to simplify the tasks of insertion and deletion, at a small price. I hope to show how Pascal simplifies the implementation of the linked, doubly-linked list.

The four tools Pascal provides for list management are pointers, which are reference types; record structures, which are user defined types; and two memory management procedures, new(ptr) and dispose(ptr). "new" allocates memory on the heap and returns a pointer to the memory, while "dispose" releases a specific piece of memory controlled by the pointer. An optional fifth tool is a memory testing function which is useful in error prevention, freect(sizeof(ptr)).

A linked list, however, is not a Pascal term or function. It is a device we will create with these four tools. A record must be defined to describe one element of the list. Included in the definition must be space to keep the link to the next item. Doubly-linked lists also have a link to the previous item. Keep in mind that you are in control of whatever "next" or "previous" means, and this makes linked lists very flexible. Items in the list may be linked in some fashion that suits the environment. Record number, for example, may be convenient for disk data. MS DOS uses a cluster number in linking and allocating disk space. For heap data, Pascal provides the pointer, which we will use. Whatever is used as a link, the program must follow the links to move from record to record. The observed effect is that the data is organized sequentially, regardless of the reality. This navigation can be performed so quickly that the response of true random access can be approached.

Let's start by setting up a doubly-linked list. To process the detail records of a single master, we need to define the detail record as it will appear in the list, and some pointers:

```
type
     variable_text = super array[1..*] of char;

     textlink = ^variable_text;

     datalink = ^detail_record;


     detail_record = record
          next_rec  = datalink;          ! ptr to next detail
          prev_rec  = datalink;          ! ptr to previous detail
          data_rec  = datarecord;        ! structured data
          text_rec  = textlink;          ! ptr to text
          end;

var
                                      ! pointers to:
     listroot  :    datalink;           ! Beginning of list
     cur_item  :    datalink;           ! Current item
```

What do we have here? Textlink and datalink are pointers, whose values will be provided by Pascal's heap management. Detail_record is the definition of an item in the list. When memory is allocated for an item to bring it into existence, next_rec and prev_rec will be loaded by our code with the proper pointers. Text_rec will likewise point to a variable length area that, like each list item, must be dynamically allocated. The free pointer listroot will anchor the list for the program, and likewise cur_item will be a convenient way to access any chosen item. Data_rec references the type data-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

record, which we assume is defined elsewhere. Besides clarity, there is another very good reason to use this technique: the record type datarecord can also be used in an I/O routine to transfer data from disk, or to print it.

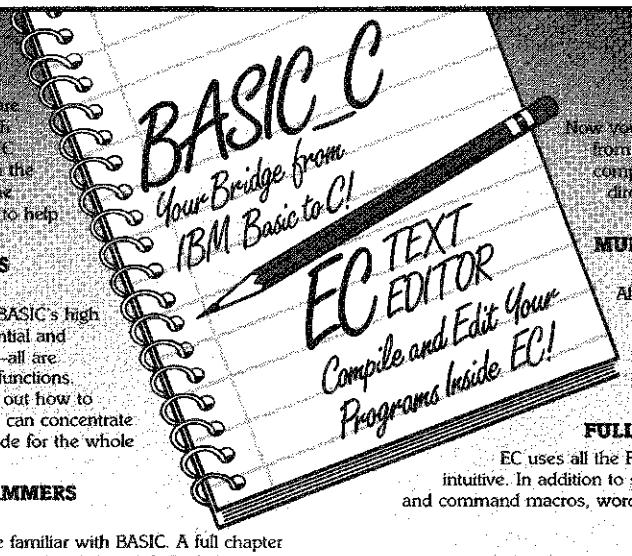
Notice that the definitions for datalink, detail_record, and next_rec are connected in a circular fashion. It is this key relationship that makes linked list processing with pointers so powerful and elegant. By using pointers as objects within records, the compiler can provide addressability to other records.

You may also have noticed that variable_text is a super array without a defined length. Pascal allows an explicit length parameter in the allocation routine, since the length of an element of variable_text may be unknown until we need to allocate the space for it.

A Pascal pointer type, by the way, is a "black box," which mostly relieves us from caring about the way it is really implemented. The structure of a pointer is intimately tied to the cpu and memory model the compiler supports, and the value is determined dynamically. Once memory is allocated and the pointer to an item in the list is determined, we can depend on it as a constant. Outside the context of the program, however, a pointer is meaningless; it would do no good to save pointers for use by another program or even the same program for another execution. Pointers, in short, are temporal.

Now let's build a linked list of detail records on the heap. The calling program passes listroot as a variable:

```
function bldlist(var root : datalink) : boolean;

var
     ptr       :    datalink;          ! pointer to new item
     end_list  :    datalink;          ! pointer to end of list
     txt_ptr   :    textlink;          ! pointer to variable text
     length    :    word;              ! length of variable text

begin     {bldlist}
     bldlist := TRUE;                  ! Assume all goes OK
     root := NIL;
     done := retrieve(input_data);     ! get detail data
     while not done do begin
                                       ! (Optional memory check)

          new(ptr);                    ! Allocate space on heap

          if (root = NIL) then ! 1st time thru
               root := ptr
          else begin                   ! link item to list
               end_list^.next_rec  := ptr;        ! forward
               ptr^.prev_rec       := end_list;   ! backward
               end;
```

BASIC_C™ LIBRARY—
BRIDGING THE BASIC GAP—

Every day, programmers like yourself are making the jump from BASIC to C. With high level functions that work like BASIC and documentation that is written from the perspective of a BASIC programmer, the BASIC_C library provides the best path to help make your journey pleasant and fast!

BASIC'S POWERFUL FUNCTIONS IN C PROGRAMS—

The BASIC_C library provides most of BASIC's high level functions—string handling, sequential and random i/o, graphics and many more—all are designed to look and work like BASIC functions. Instead of wasting time trying to figure out how to clear a screen or locate the cursor, you can concentrate on learning what C is about. Source code for the whole library is included!

WRITTEN WITH BASIC PROGRAMMERS IN MIND—

Our documentation is oriented to those familiar with BASIC. A full chapter is devoted to a point-by-point comparison of BASIC and C. Each function's description contains at least one example written in BASIC and the equivalent C. Many customers tell us that the BASIC_C documentation is the best C tutorial they've seen.

EC DEMO DISK -	$ 5
EC (pronounced e-see) - full screen editor	$125
BASIC_C LIBRARY - including source code	$175
BASIC_C & EC	$275
BASIC_C & DeSmet C compiler/w debugger	$325
BASIC_C & Computer Innovations C compiler	$475
BASIC_C & Lattice C compiler	$525
BASIC_C & EC & DeSmet C compiler/w debugger	$425
BASIC_C & EC & Computer Innovations C compiler	$575
BASIC_C & EC & Lattice C compiler	$625

EC TEXT EDITOR—
MAKE ANY DOS CALL FROM INSIDE EC—

Now you can edit, compile and test your programs from inside the EC editor. In fact, EC gives you complete access to the operating system. Do a directory, copy or delete files, even run other programs without ever leaving EC!

MULTIPLE WINDOWS ... WITH A FILE IN EACH—

All windows can be shown on the screen at the same time, or the screen can be dedicated to one file, while the others are kept in the background—only a keystroke away. You can even cut and paste blocks of text between windows!

FULL SCREEN EDITOR FOR THE IBM—

EC uses all the PC's special keys so that editing is fast and intuitive. In addition to standard editing features, EC provides text and command macros, word wrap, paragraph reformatting, horizontal scrolling, and color video control.

DEMO DISK IS ONLY $5—

Call or write for our full-featured demo. You'll get a standard version of EC that can handle up to 10K worth of text files and a complete set of documentation. See for yourself how pleasant it is to use an editor that gives you multiple windows and free access to your operating system.

*BASIC_C and EC run on an IBM PC, or look-alike, with at least 128K RAM under DOS 1.1 or higher. To use the EC EDITOR'S DOS interface feature you must have DOS 2.0 or higher and enough RAM to run the additional program.

IBM is a trademark of International Business Machines.
MO. Residents Add 6% Sales Tax.

SOURCE

C SOURCE
12801 Frost Road
Kansas City, MO 64138

816-353-8808

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        end_list := ptr;                    ! the new end of list
        ptr^.next_rec := root;              ! last wraps to 1st
        root^.prev_rec := ptr;              ! 1st wraps to last


                    ! Now, copy the input data to list
        ptr^.data_rec := input_data^.data_rec;

        ptr^.text_rec := NIL;               ! No text yet

        done := retrieve(input_data);       ! get more data
        end;
end; {bldlist}
```

This routine illustrates how memory is reserved for a single item on the list, and the housekeeping required to connect this item to the list that exists to this point. Each item is simply added to the end of the list by the way the next_rec and prev_rec pointers are set. By using the pointer values in listroot and end_list, the list is made circular. Note that after an item is added to the list, the record defined by data_rec is copied into the list. A better approach would be to put this process outside this function; we leave it here to illustrate a data movement technique. Text_rec is set NIL to show the safe way to deal with pointers that may be established later; when the new(ptr) function is executed, storage is not initialized, and the garbage in text_rec might just as well be a valid pointer after we leave this routine.

Note also that the function retrieve(input_data), which returns TRUE or FALSE, is left to your imagination.

To tidy this routine up, we add a memory checking function to ensure that enough space is left on the heap to add another item to the list. This function, however, is not available in IBM Pascal 1.0.

```
Replace --

                                    ! (Optional memory check)
With --
        if (freect(sizeof(ptr)) <= 2) then begin
            bldlist := FALSE;               ! not enuf room for 1
            done    := TRUE;                !  item + slop in list
            end
        else begin

            - - -

            end;

Finally, there is text_rec to take care of, which was left
NIL:

        length := gettext;              ! get optional text that
                                        !  this record may have
        if (length > 0) and
           (freect(length) >= 1) then begin
            new(txt_ptr, ord(length));      ! Alloc exact memory
            ptr^.text_rec := txt_ptr;       ! Save the pointer
                                            ! Copy text to list
            movel (adr text_in[1], adr txt_ptr^[1], length);
            end;
```

The text is being placed in the list defined by variable_text, which is linked to the detail record list by placing the text pointer into the detail record. Only the amount of storage required to contain the text is committed; if editing the text requires more storage, it is efficient to commit new storage for the edited text, dispose of the old storage, and change the pointer.

By this point you may see that although we are dealing with a doubly-linked circular list as well as a subordinate list with variable length elements, the code to build this — with error checking to boot — is not particularly complex. Maintaining the lists is not complex either.

Let's proceed, then, and add routines to insert and delete items from the doubly-linked list. To insert an item, we assume that the main program has placed the value of the pointer to the current item in curr. This is passed as a parameter to the function insitem, and will be updated to point to the new item if the insert is successful:

```
function insitem(var curr:datalink):boolean;
var
    ptr  :    datalink;

begin {insitem}
    if (freect(sizeof(ptr)) <= 2) then
        insitem := FALSE                ! no space left
    else begin
        new(ptr);                       ! Allocate space

                        ! Now insert item into list following the
                        ! current item
        ptr^.next_rec := curr^.next_rec;      ! Forward
        ptr^.prev_rec := curr;                ! Backward

                        ! Connect the list to the new item
        curr^.next_rec := ptr;
        ptr^.next_rec^.prev_rec := ptr;

        ptr^.text_rec := NIL;           ! Init text pointer

        curr := ptr;                    ! New item is now current

        insitem := TRUE;                ! successful insert
        end;
end;  {insitem}
```

This routine illustrates how straightforward it is to maintain a doubly-linked list. Notice that if we happen to insert after the last item in the list, circularity will be automatically maintained. Also notice that two levels of dereference are used to update the previous record pointer of the next record when we connect the list to the new item. You will find the ability to dereference to a known depth very handy at times. Dereference, by the way, is an important term which has unfortunately not found its way into many glossaries. If you think of it as a type of indirect addressing, it may make more sense when compiler error 209 "Cannot Dereference Value" rears its ugly head.

Deleting an entry from our list requires input of the listroot value and the current item pointer. The pointer value returned is usually only to the current item, which is the item previous

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

to that which was deleted. If the root item is deleted, the root pointer is also modified. And if the only item in the list is deleted, FALSE is returned.

```
function delitem(var root, curr: datalink):boolean;
var
     pptr :    datalink;                ! ptr to previous item

begin {delitem}
     pptr := curr^.next_rec;        ! return next item as new
                                    !  current pointer

                          ! Self-healing removal of current item
                          ! from the list
     curr^.prev_rec^.next_rec := curr^.next_rec;          ! Forward
     curr^.next_rec^.prev_rec := curr^.prev_rec;          ! Backward

     if (curr = root) then
          root := curr^.next_rec;          ! Deleted root

     if (curr^.prev_rec = curr) then
          root := NIL;                     ! Deleted only item in lst

     if (curr^.text_rec <> NIL) then
          dispose(curr^.text_rec);         ! Release text storage

     dispose(curr);                        ! Release record storage

     if (root <> NIL) then begin
          curr := pptr;                    ! return current pointer
          delitem := TRUE;
          end
     else begin
          curr := NIL;
          delitem := FALSE;           ! list is all deleted
          end;

end; {delitem}
```

Multi-level dereferencing is used again to remove the item from the list. Special processing is required if the root item or the only item in the list is deleted, since we need to assure the integrity of the root pointer. The dispose() function is required to reclaim precious space in the heap, which may never exceed 64k in MS Pascal. It is excellent practice to release unwanted storage anyway, and the cost is small: The delitem routine, for example, can be called from a tight loop that starts anywhere in the list and deletes until FALSE is returned. Even with several hundred items in the list, nearly sub-second response time should be experienced.

The insert and delete functions may be used together to provide a move function. Prior to entering the move function, assume that the source and destination pointers have been determined:

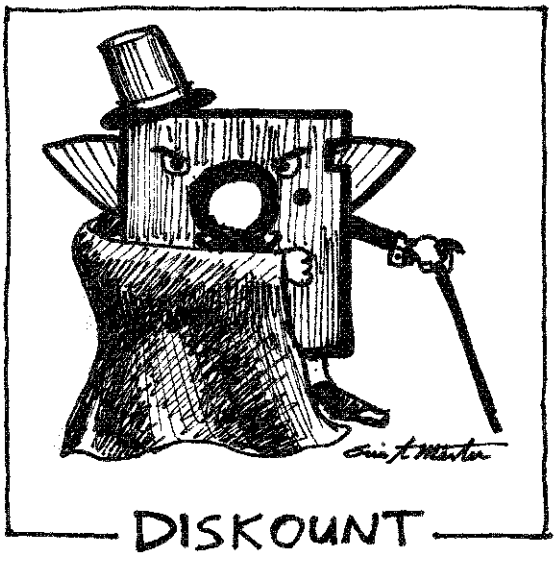
```
function movitem (var root, source, dest : datalink):boolean;
var
     junk :    boolean;

begin {movitem}
     if (insitem(dest)) then begin
          ptr^.data_rec := source^.data_rec; ! Copy the record
          ptr^.text_rec := source^.text_rec; ! Copy text link
          junk := delitem(root, source);     ! delete can't fail
          movitem := TRUE;
          end
     else
          movitem := FALSE;
end; {movitem}
```

When the move function completes, dest will point to the moved item, source will point to the next item past the moved item, and root will usually be unchanged. Thus, repeated calls to movitem can effect a block move.

This discussion has so far covered doubly-linked circular lists, with a singly-linked variable-length list added for good measure. Along the way, we have seen how to ensure that enough memory exists in the heap to avoid crashing, how to ensure pointers stay valid, how to use pointers as parameters and multi-level references, and how to release storage back to the heap in an orderly manner.

The great flexibility of lists means that the examples above can be implemented with other alternatives. For instance, the record structure data_rec could be just a pointer to the record. As a starting point, I would encourage you to explore the following questions: What are the advantages and disadvantages of this alternative? How can the basic illustrations be expanded to manage more than one list to the data we wish to process? What are the uses for more than one list to the same data? What are some of the considerations in maintaining these lists? Can the movitem function be performed just by switching pointers? What are the advantages and disadvantages?

One last item should be discussed, and that is an apparent bug in MS Pascal 3.13 that will loop PAS2. It actually does not seriously impact pointers. The problem was observed in an



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

unusual situation: subscripting an array with a pointer to a subscripted super array of records:

```
if(ARRAY1[ptr^[i].start].FIELD <> ARRAY1[ptr^[i].end].FIELD) then

This was decomposed into:
    j := ptr^[i].start;
    k := ptr^[i].end;
    if(ARRAY1[j].FIELD <> ARRAY[k].FIELD) then ...
```

Pointers are a very powerful feature of Pascal. They are implemented at so fundamental a level of the language that you may use them virtually anywhere you would use an ordinary variable. They can be used within subscripts, allowing the paragraph above as an exception, saving the trouble of defining and loading an integer variable needed for a subscript. The one caveat is that IBM 1.0 Pascal seems to be afflicted with some undocumented shortfalls in this area — since switching to 3.13 early this year a lot of little problems seemed to just vanish. Now that IBM has released 2.0, the picture is much brighter for all 1.0 owners.

If you have not used pointers for linked lists, this introduction may encourage you to try them. Thanks to the elegant manner in which Pascal implements the basic tools of list processing, fast and powerful routines can be coded to exploit the flexibility of linked lists.

Additional reading: One text that excells in the depth and breadth of list processing is "Personal Pascal" by David E. Cortesi and George W. Cherry (Reston, 1984). This book is excellent throughout, incorporating the extensions of Microsoft's and Digital Research's Pascal compilers into the discussions and examples. □

MISC.

KING FLOPPY KONG

Eastman Kodak's new floppy drive has been picked up by a number of companies. Data Technology is selling a series of Winchester disk drives (half-height) with a Kodak (half-height) floppy in one small box. The Kodak floppy holds 2.78 megabytes of storage. Obviously a convenient way to backup your main storage device, these are a very compatible pair. Actually the Kodak floppy drive will hold as much as 3.3 megabytes.

TeamMate 1000 (10 meg plus Kodak Floppy)
Data Technology
2775 Northwestern Parkway
Santa Clara, CA 95051
(408) 496-0434

CREATE DIFFERENT DISKETTE FORMATS USING YOUR PC

If you need many different kinds of diskettes duplicated, you should know that Vertex Systems has a software package that will read, write, format, and duplicate nearly 60 different types of formats. Disk formats include the most popular CP/M computers plus CP/M-86, TRS-DOS 1.3, p-System, NECDOS, and European types. Both 40- and 80-track CP/M formats are covered. An ASCII conversion utility makes processing text files easier.

Vertex also sells a CP/M-80 emulator (80MATE) that simulates a CP/M 2.2 environment running under MS-DOS.

Xeno-Disk Copier ($379.50)
80MATE ($99.50)
Vertex Systems
7950 W. 4th St.
Los Angeles, CA 90048
(213) 938-0857

TURBO PASCAL COPIES

Borland is offering an educational version of their Turbo Pascal for schools. Once the package is purchased, a school may copy it as many times as needed within the institution. Fifteen user manuals are provided; additional ones cost $15.

Alex Morton
Borland International
4113 Scotts Valley Dr.
Scotts Valley, CA 95066
(408) 438-8400

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MISC.

SMARTCOM UPDATE

Hayes Microcomputer has announced a new version of their Smartcom II communications package for the IBM PC. They finally included the XMODEM protocol so we don't have to switch back and forth between PC-TALK to download EXE files. Hayes's update should be out by the time you read this. You can upgrade your current version, but they have not decided on the fee yet.

Smartcom II Version 2.0
Hayes Microcomputer Products
5923 Peachtree Industrial Blvd.
Norcross, GA 30092
(404) 449-8791

LISA-BASICA

Pterodactyl Software is supporting a PC BASIC compiler for the Apple Lisa computer which is compatible with IBM's BASICA compiler. BASIC programs can be quickly converted to run on a Lisa. Links are provided to Lisa's advanced graphics features.

PC BASIC sells for $1000, which includes a compiler, communications package for transporting data files from an IBM PC to a Lisa, a one-time-only license fee of $750, plus a money back guarantee. Companies can have their programs converted by Pterodactyl Software for $2500. Programs larger than 500K or containing PEEKS, POKES, or assembler subroutines will cost more. For more information contact:

Craig Settles
PC BASIC
SUCCESS
1755 Le Roy Ave.
Suite 214
Berkeley, CA 94709
(415) 644-1154

INFORMATION MANAGER : DB-FABS

Computer Control Systems has announced DB-FABS, a general purpose data, screen and report manager written in 8088 assembly language. Readily usable by both novices and professional computer users. Currently available for Microsoft BASIC, both interpreted and compiled. Create and manage files, screens, reports, and output forms.

DB-FAB's main module is a high-speed, keyed access, BTREE subroutine. AUTOSORT is an additional module that will sort/select/merge records. It comes as a subroutine to be included in a larger program or as a stand-alone sorting utility.

DB-FABS ($295)
Computer Control Systems
298 21st Terrace S.E.
Largo, FL 33541
(813) 586-1886

PASCAL ON A PLATE

Gamma Software has developed a Pascal Tutor program for disk based instruction. It teaches you the fundamentals of Pascal and is intended for users who have had at least some experience with other computer languages. More than 500 text screens are presented with more than 150 questions and answers. 34 program examples are discussed. The principal topics are covered — scalar data types, control structures, procedures, functions, arrays, records, files, sets and pointers. Pascal Tutor can be run in text or quiz mode.

Pascal Tutor ($59.95)
Gamma Software
P. O. Box 25625
Los Angeles, CA 90025
(213) 473-7441

PC-WRITE VERSION 2.0 UPDATE

The new release of PC-WRITE is now available. Everything from extended keyboard redefinition to DOS 2.0 pathnames and the ability to execute commands from the editor is included. The top line shows the mode you are in, which means you have 24 lines of free space for full-screen editing. More printers are installable, reformat paragraph enhancements have been added, WordStar control keys are customizable, and there is a newer help-screen layout. If you can believe it, PC-WRITE is even faster than before. Undoubtedly the best $10 you could spend on a word processor. (And source code, in both assembly and Pascal, is $75.)

PC-WRITE ($10)
Quicksoft
219 First N., #224
Seattle, WA 98109
(206) 282-0452 □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TOOLS, Hybridizing Assembly and BASIC Compiler Routines

Corrected Code listing from Vol 2 No.3

Article by Wat Buchanon, President, The FreeSoft Company

If you don't have the Macro Assembler, program Listing #1 below will allow you to create the DISKIO.OBJ module from BASIC.

```
  5 REM ** Don't forget to enter BASIC using the command: BASIC/S:95
 10 REM ** Program Listing #1, This creates the assembler object module
 20 REM ** DISKIO.OBJ for those of you who don't have the Macro
    Assembler.
 30 FOR Z=1 TO 95:READ A:A$=A$+CHR$(A):T=T+A:NEXT Z
 40 IF T<>6400 THEN PRINT "Whoops!  At least one of the numbers in
    the DATA statements":PRINT"has been typed in wrong.  Check your
    listing again!":END
 50 INPUT "Drive for DISKIO.OBJ (don't enter a colon) ";DR$
 60 OPEN "R",1,DR$+":"+"DISKIO.OBJ",95:FIELD 1,95 AS K$:LSET K$=A$
 65 PUT 1,1:CLOSE
 70 PRINT:PRINT"Program DISKIO.OBJ has been created!":END
 80 DATA 128,3,0,1,65,59,150,7,0,0,4,67,79,68,69,68,152,7,0,96
 90 DATA 36,0,2,2,1,216,160,40,0,1,0,0,85,139,236,139,118,12,139,4
100 DATA 139,118,10,139,28,139,118,8,139,12,139,118,6,139,20,205
110 DATA 19,208,208,139,118,12,137,4,93,202,8,0,142,144,13,0,0,1
120 DATA 6,68,73,83,75,73,79,0,0,0,153,138,7,0,193,0,1,1,0,0,172
```

Now we need some BASIC code that will take care of the keyboard and display routines, as well as pass the values to the DISKIO module we just created. Here is program #2, DISKREAD.BAS:

```
 10 REM ** Program Listing #2, Accessing a raw disk sector through a
    BIOS call
 20 REM ** from BASIC Compiler source code.
 25 DEFINT A-Z:DIM BUFFER1%(512)
 30 INPUT "Lowest byte value to print (0 to 254) ";LB:IF LB<0 OR
    LB>255 GOTO 30
 35 PRINT "Highest byte value to print (";LB+1;" to 255) ";:INPUT HB
 37 IF HB<LB+1 OR HB>255 GOTO 35
 40 INPUT "Drive (A=0, B=1, C=3, D=4) ";DR
 50 IF DR<0 OR DR>4 GOTO 40
 60 INPUT "Side (0 or 1) ";H
 70 IF H<>0 AND H<>1 GOTO 60
 80 INPUT "Track (0 to 39) ";T
 90 IF T<0 OR T>39 GOTO 80
100 INPUT "Sector (1 to 9) ";S
110 IF S<1 OR S>9 GOTO 100
120 REM ** Lines 40 to 100 get values needed by BIOS call.  You can
    allow track
130 REM ** reads higher than track 39 by changing line 90.  Similarly,
    the
140 REM ** range of sectors allowed in line 110 is purely arbitrary.
150 A=0:B=0:C=0:D=0:CALL DISKIO(A,B,C,D)
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
160 REM ** Line 150 just resets the drives and prepares for BIOS calls.
170 A=2*256+1:REM ** This is the sector read instruction
180 B=VARPTR(BUFFER1%(0)):REM ** This is where the sector will go into
    memory.
190 C=T*256+S:REM ** This is the track and sector values to be passed
200 D=H*256+DR:REM ** This is the side and drive values to be passed
210 CALL DISKIO(A,B,C,D):REM ** Call the assembly routine passing
    values.
220 REM ** Now let's extract the value passed back from the assembly
    routine
230 REM ** in variable A that tells us whether our read attempt was
    successful
240 REM ** or not, and act accordingly.
250 AH=INT(A/256):AL=A-(AH*256):AH=ABS(AH):IF (AL AND 1)=0 GOTO 500
260 REM ** If read was successful, we branch directly to line #500.
270 REM ** Otherwise, we should print out what sort of error occurred.
280 IF AH=2 THEN A$="Address mark not found"
290 IF AH=3 THEN A$="Disk is write protected"
300 IF AH=4 THEN A$="Sector not found"
310 IF AH=8 THEN A$="Direct memory access overrun"
320 IF AH=9 THEN A$="Direct memory access crossed a 64K boundary"
330 IF AH=16 THEN A$="Cyclic redundency check (CRC) error"
340 IF AH=32 THEN A$="Disk controller circuit failure"
350 IF AH=64 THEN A$="Track not found"
360 IF AH=128 THEN A$="Drive not ready. (Door open?  Disk mounted?)"
370 CLS:PRINT"ERROR!  The following happened during the disk read:"
380 PRINT"---> ";A$;" <---"
390 INPUT "Re-try read? (Y or N) ";A$:B$=LEFT$(A$,1)
400 IF B$="Y" OR B$="y" GOTO 150
410 IF B$="N" OR B$="n" GOTO 600
420 REM ** Line 500 is where execution branchs to after a successful
    read.
430 REM ** The actual sector data is held array BUFFER1%.  Since this
    is
440 REM ** an integer array (2 bytes for each element), it's easier to
450 REM ** do PEEKS using the VARPTR command than try to decode the
    individual
460 REM ** array elements into most significant and least significant
    bytes.
470 REM ** You can do whatever you like to the sector data from this
    point,
480 REM ** below we'll just print out the ASCII equivalent of the
    sector bytes.
500 K=VARPTR(BUFFER1%(0)):REM ** Find memory location of array.
510 CLS:FOR Z=K TO K+511:REM ** Print out 512 bytes (room in array
    for 1024).
520 K$=CHR$(PEEK(Z)):IF ASC(K$)<LB OR ASC(K$)>HB THEN PRINT "."; ELSE
    PRINT K$;
530 NEXT Z:REM ** Continue loop to end of sector
600 PRINT:INPUT "Read another sector? (Y or N)";A$:B$=LEFT$(A$,1)
610 IF A$="Y" OR A$="y" THEN RUN ELSE END
```

Type in the above program and save it in ASCII (uncompressed) format using the BASIC command:

```
SAVE "DISKREAD.BAS",A
```

□

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ARE YOU AN ISSUE SHY OF A FULL SET?

Complete your library! Order any back issues you may have missed — or become a new subscriber. Use the form below or the handy subscription envelope provided. You'll be glad you did.

- Vol. 1, No. 1: Norton Pascal I/O, Program Style Manual, COBOL, System Analysis, BASIC modules, Rounding error.
- Vol. 1, No. 2: Defining character sets, C86 BIOS calls, Copy protection, Pascal code, Input routines, Writing an Assembler.
- Vol. 1, No. 3: ASM dumb terminal, C source code, ASM graphics line drawing, Text editors reviewed, Utilities.
- Vol. 1, No. 4: Diskette directory via Pascal, MASM WHILE-WEND, C/ASM interface, ASM stay resident routine, Structured programming.
- Vol. 1, No. 5: Multi-tasking software review, Transporting C Programs, Programmers calculator, dBase bargraphs, Utilities.
- Vol. 1, No. 6: ASM printer configuration, DOS patch utility, Hi-speed graphics, Software ownership, dWINDOW review.
- Vol. 2, No. 1: Freelance consulting, Sorts in C, Pascal text editor source code, Designing national database, Ultra utilities.
- Vol. 2, No. 2: Understanding C through Assembly language, Utility routines in C and Assembly, Music in C (using a C music program), Cornucopia of software development tools.
- Vol. 2, No. 3: Software development environments, Creating a productive screen handling tool, Generating graphics within windows, Create, edit and play your musical ideas, Linking Assembler to Pascal and Basic.
- Vol. 2, No. 4: Writing a custom ROM BIOS for PC-DOS, The future of disk protection, Time-saving MASM Macros, C routines you can purchase.

Name _____

Company _____

Address _____

City _____ State _____ Zip _____

Telephone _____

Send me the following back issues:
($4 each U.S., $5 Canada, $7 Other, U.S. funds)

_____ Vol. 1, No. 1
_____ Vol. 1, No. 2
_____ Vol. 1, No. 3
_____ Vol. 1, No. 4
_____ Vol. 1, No. 5
_____ Vol. 1, No. 6
_____ Vol. 2, No. 1
_____ Vol. 2, No. 2
_____ Vol. 2, No. 3
_____ Vol. 2, No. 4

☐ 1 year bi-monthly subscription:
$24.00 ($30 Canada, $40 other, U.S. funds only)

☐ Please renew my subscription

☐ Check enclosed ☐ Bill me

☐ Charge my Bankcard: ☐ VISA ☐ MC ☐ Am Ex

_____ Expir. Date _____

Signature _____

Send to:
PROGRAMMER'S JOURNAL
P.O. Box 30160
Eugene, OR 97403
(503) 345-3043

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

IN THIS ISSUE

68000 vs. 80286

CORE Executive

PC/IX & Unix Support

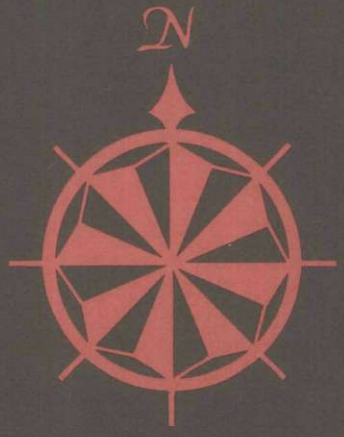
The C Bandwagon

Screen I/O Routines

A PC-Resident Programmer's Toolkit

Assembly Routines from dBASE II

Pascal's Pointer Power



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED